



Module M53

Partha Pratim
Das

Objectives &
Outlines

λ in C++:
Recap

`std::function`

Examples

Generic λ

Recursive λ in
C++

Practice Examples

Generic Recursive λ

Practice Examples

Generalized λ
Captures

Module Summary

Programming in Modern C++

Module M53: C++11 and beyond: General Features: Part 8: Lambda in C++/2

Partha Pratim Das

Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur

ppd@cse.iitkgp.ac.in

All url's in this module have been accessed in September, 2021 and found to be functional



Module Recap

Module M53

Partha Pratim
Das

Objectives &
Outlines

λ in C++:
Recap

`std::function`

Examples

Generic λ

Recursive λ in
C++

Practice Examples

Generic Recursive λ

Practice Examples

Generalized λ
Captures

Module Summary

- Understood λ expressions (unnamed function objects) in C++ with
 - Closure Objects
 - Parameters
 - Capture



Module Objectives

Module M53

Partha Pratim
Das

Objectives & Outlines

λ in C++:
Recap

`std::function`

Examples

Generic λ

Recursive λ in
C++

Practice Examples

Generic Recursive λ

Practice Examples

Generalized λ
Captures

Module Summary

- To learn different techniques for writing and using λ expressions in C++
- To understand `std::function` for specifying function prototypes
- To understand generic λ support in C++14
- To learn how to implement recursive λ expressions in C++11 and C++14
- To be exposed to several practice examples



Module Outline

Module M53

Partha Pratim
Das

Objectives & Outlines

λ in C++:
Recap

`std::function`

Examples

Generic λ

Recursive λ in
C++

Practice Examples

Generic Recursive λ

Practice Examples

Generalized λ
Captures

Module Summary

- 1 λ in C++: Recap
- 2 `std::function`
 - Examples
- 3 Generic λ in C++14
- 4 Recursive λ in C++
 - Practice Examples
 - Generic Recursive λ
 - Practice Examples
- 5 Generalized λ Captures
- 6 Module Summary



λ in C++: Recap

Module M53

Partha Pratim
Das

Objectives &
Outlines

λ in C++:
Recap

`std::function`

Examples

Generic λ

Recursive λ in
C++

Practice Examples

Generic Recursive λ

Practice Examples

Generalized λ
Captures

Module Summary

λ in C++: Recap

Source:

- Module 52: C++11 and beyond: General Features: Part 7: Lambda in C++/1



λ in C++: Recap (Module 52)

Module M53

Partha Pratim Das

Objectives & Outlines

λ in C++:
Recap

std::function

Examples

Generic λ

Recursive λ in C++

Practice Examples

Generic Recursive λ

Practice Examples

Generalized λ Captures

Module Summary

- A λ *expression* is an unnamed function for specifying lightweight functions in C++
- Compiler generates a functor-like class for a λ which at run-time instantiates a *Closure Object*
- A λ *expression* can have zero or more value and / or reference *parameters* used in its body
- Free variables in the body a λ *expression* must be *captured* by value or by reference
- Here is a complete example

```
class A { std::vector<int> values; int m_;
public: A(int mod) : m_(mod) { }
    A& put(int v) { values.push_back(v); return *this; }
    int extras() { int count = 0;
        std::for_each(values.begin(), values.end(), // iterate over values to apply lambda
            [=, &count] // capture default by value, capture count by reference
            (int v) { count += v % m_; }); // v is a param, this captured implicitly
        // by value & m_ used as this->m_ to accumulate

        return count;
    }
};

int main() { A g(4);          // g.m_ = 4
    g.put(3).put(7).put(8); // g.values = { 3, 7, 8 }
    std::cout << "extras: " << g.extras(); // extras: 6: 3%4 + 7%4 + 8%4 = 3+3+0 = 6
}
```

Programming in Modern C++



std::function

Module M53

Partha Pratim
Das

Objectives &
Outlines

λ in C++:
Recap

std::function

Examples

Generic λ

Recursive λ in
C++

Practice Examples

Generic Recursive λ

Practice Examples

Generalized λ
Captures

Module Summary

std::function

Source:

- [std::function](http://en.cppreference.com), [cppreference.com](http://en.cppreference.com)
- [Usage and Syntax of std::function](http://stackoverflow.com), stackoverflow.com
- [std::function](http://en.cppreference.com), [cplusplus.com](http://en.cppreference.com)
- [std::function::function](http://en.cppreference.com), [cplusplus.com](http://en.cppreference.com)



std::function

Module M53

Partha Pratim Das

Objectives & Outlines

λ in C++:
Recap

std::function

Examples

Generic λ

Recursive λ in C++

Practice Examples

Generic Recursive λ

Practice Examples

Generalized λ
Captures

Module Summary

- `std::function` is defined in `<functional>`

```
template <class T> function;           // empty - undefined
template <class Ret, class... Args> class function<Ret(Args...)>;
```
- It is a polymorphic wrapper for function objects applies to all callable elements:
 - Function pointers
 - Member function pointers
 - Functors (including closure objects)

Type	Old School Define	std::function
Free	<code>int(*callback)(int,int)</code>	<code>function< int(int,int) ></code>
Functor	<code>object_t callback</code>	<code>function< int(int,int) ></code>
Member	<code>int (object_t::*callback)(int,int)</code>	<code>function< int(int,int) ></code>

- Function declarator syntax is: `std::function< R (A1, A2, A3...) > f;`
- The function object can be copied and moved around, and can be used to directly invoke the callable object with the specified call signature
- The function objects can also be in a state with no target callable object, called *empty functions*. Calling them throws a `bad_function_call` exception



std::function: Examples

Module M53

Partha Pratim Das

Objectives & Outlines

λ in C++:
Recap

std::function

Examples

Generic λ

Recursive λ in C++

Practice Examples

Generic Recursive λ

Practice Examples

Generalized λ Captures

Module Summary

```
#include <iostream>           // std::cout
#include <functional>         // std::function, std::negate

int half(int x) { return x/2; } // a function
struct third_t { int operator()(int x) { return x/3; } }; // a function object class
struct MyValue { int value; int fifth() { return value/5; } }; // a class with data members

int main () {
    std::function<int(int)> fn1 = half; // function
    std::function<int(int)> fn2 = &half; // function pointer
    std::function<int(int)> fn3 = third_t(); // function object
    std::function<int(int)> fn4 = [](int x) { return x/4; }; // lambda expression
    std::function<int(int)> fn5 = std::negate<int>(); // standard function object

    std::cout << "fn1(60): " << fn1(60) << '\n'; // fn1(60): 30
    std::cout << "fn2(60): " << fn2(60) << '\n'; // fn2(60): 30
    std::cout << "fn3(60): " << fn3(60) << '\n'; // fn3(60): 20
    std::cout << "fn4(60): " << fn4(60) << '\n'; // fn4(60): 15
    std::cout << "fn5(60): " << fn5(60) << '\n'; // fn5(60): -60

    std::function<int(MyValue&)> value = &MyValue::value; // pointer to data member
    std::function<int(MyValue&)> fifth = &MyValue::fifth; // pointer to member function
    MyValue sixty {60}; std::cout << "value(sixty): " << value(sixty) << '\n'; // value(sixty): 60
    std::cout << "fifth(sixty): " << fifth(sixty) << '\n'; // fifth(sixty): 12
}
```



std::function: Example: Pipeline

Module M53

Partha Pratim Das

Objectives & Outlines

λ in C++:
Recap

std::function

Examples

Generic λ

Recursive λ in C++

Practice Examples

Generic Recursive λ

Practice Examples

Generalized λ
Captures

Module Summary

```

#include <iostream>    // std::cout
#include <algorithm>    // std::for_each
#include <vector>       // std::vector
#include <functional>   // std::function

struct machine {
    template< typename T >
    void add(T f) { to_do.push_back(f); } // adding a function to to_do pipeline of functions
    int run(int v) {
        std::for_each(to_do.begin(), to_do.end(), // iterate over the vector of pipeline of functions
            [&v](std::function<int(int)> f) {
                v = f(v); // apply the next function and pipeline the result
            });
        return v;
    }
};

std::vector< std::function<int(int)> > to_do; // to_do is a vector collection of pipeline functions

int foo(int i) { return i + 4; }

int main() { machine m;
    m.add([](int i){ return i * 3; }); // add a lambda as function #1 in pipeline
    m.add(foo);                       // add foo as function #2 in pipeline
    m.add([](int i){ return i / 5; }); // add a lambda as function #3 in pipeline
    std::cout << "run(7) : " << m.run(7) << std::endl;
}

run(7) : 5 // func. #1 on 7 => 7 * 3 = 21. func. #2 on 21 => 21 + 4 = 25. func. #3 on 25 => 25 / 5 = 5

```



Generic λ in C++14

Module M53

Partha Pratim Das

Objectives & Outlines

λ in C++:
Recap

`std::function`

Examples

Generic λ

Recursive λ in C++

Practice Examples

Generic Recursive λ

Practice Examples

Generalized λ
Captures

Module Summary

Source:

- [Generic lambdas, ISO](#)
- [Lambda expressions \(since C++11\)](#), [cppreference.com](#)
- [Scott Meyers on C++](#)
- [Lambda expressions in C++, Microsoft](#)
- [Generalized Lambda Expressions in C++14](#)
- [Generic code with generic lambda expression](#)

Generic λ in C++14



Generic / Generalized λ in C++14

Module M53

Partha Pratim Das

Objectives & Outlines

λ in C++:
Recap

`std::function`

Examples

Generic λ

Recursive λ in C++

Practice Examples

Generic Recursive λ

Practice Examples

Generalized λ
Captures

Module Summary

- C++11 introduced λ expressions as short inline anonymous functions that can be nested inside other functions and function calls

- C++ 14 buffed up λ expressions by introducing Generic or Generalized λ

- Following λ function returns the sum of two integers

```
[<](int a, int b) -> int { return a + b; } // Return type is optional
```

- Whereas we need a different λ to obtain the sum of two floating point values:

```
[<](double a, double b) -> double { return a + b; } // Return type is optional
```

- In C++11 we could unify these two λ functions using template parameters:

```
template<typename T>  
[<](T a, T b) -> T { return a + b } // Return type is optional - compiler may infer
```

- C++ 14 circumvent this by the keyword `auto` in the input parameters of the λ expression. Thus the compilers can now deduce the type of parameters during compile time:

```
[<](auto a, auto b) { return a + b; } // Compiler must infer return type
```



Generic / Generalized λ in C++14

Module M53

Partha Pratim Das

Objectives & Outlines

λ in C++:
Recap

std::function

Examples

Generic λ

Recursive λ in C++

Practice Examples

Generic Recursive λ

Practice Examples

Generalized λ
Captures

Module Summary

```
#include <iostream>, #include <string>, using namespace std;
```

```
// C++11 lambda's - separate lambda for every type. Return type is optional
```

```
auto add_i = [](int a, int b) { return a + b; }; // Compiler may infer return type
```

```
auto add_d = [](double a, double b) { return a + b; }; // Compiler may infer return type
```

```
auto add_s = [](string a, string b) { return a + b; }; // Compiler may infer return type
```

```
// C++11 templated lambda - one lambda for multiple types. Return type is optional
```

```
template<typename T> auto add_t = [](T a, T b) { return a + b; }; // Compiler may infer return type
```

```
// C++14 generic lambda. Return type cannot be specified
```

```
auto add = [](auto a, auto b) { return a + b; }; // Compiler must infer return type
```

```
int main () {
```

```
    // Different name of each lambda for each type: No inference
```

```
    cout << add_i(3, 5); // add_i for int type // 8
```

```
    cout << add_d(2.6, 1.3); // add_d for double type // 3.9
```

```
    cout << add_s("Good ", "Day"); // add_s for string type converts from const char* // Good Day
```

```
// Same name of the lambda for all types, type must be specified: No inference
```

```
cout << add_t<int>(3, 5); // add_t<int> for int type
```

```
cout << add_t<double>(2.6, 1.3); // add_t<double> for double type
```

```
cout << add_t<string>("Good ", "Day"); // add_t<string> for string type converts from const char*
```

```
// Same name of the lambda for all types and no type need to be specified: It is inferred
```

```
cout << add(3, 5); // add for int type
```

```
cout << add(2.6, 1.3); // add for double type
```

```
cout << add(string("Good "), string("Day")); // add for string type - cannot convert from const char*
```



Return Type of Generic λ in C++14

Module M53

Partha Pratim Das

Objectives & Outlines

λ in C++:
Recap

std::function

Examples

Generic λ

Recursive λ in C++

Practice Examples

Generic Recursive λ

Practice Examples

Generalized λ Captures

Module Summary

- The return type may be inferred by the compiler from the return expression:

```
auto add = [](auto a, auto b) { return a + b; };  
cout << add(2, 3);           // 5 // rt = int  
cout << add(4, 3.3);         // 7.3 // rt = double  
cout << add(2.6, 2);          // 4.6 // rt = double  
cout << add(3.8, 4.5);        // 8.3 // rt = double
```

- The return type may be specified from the parameters:

```
auto add = [](auto a, auto b) -> decltype(a) { return a + b; };  
cout << add(2, 3);           // 5 // rt = int  
cout << add(4, 3.3);         // 7 // rt = int  
cout << add(2.6, 2);          // 4.6 // rt = double  
cout << add(3.8, 4.5);        // 8.3 // rt = double
```

```
auto add = [](auto a, auto b) -> decltype(b) { return a + b; };  
cout << add(2, 3);           // 5 // rt = int  
cout << add(4, 3.3);         // 7.3 // rt = double  
cout << add(2.6, 2);          // 4 // rt = int  
cout << add(3.8, 4.5);        // 8.3 // rt = double
```

- The return type may be explicitly specified:

```
auto add = [](auto a, auto b) -> int { return a + b; };
```



Use of Generic λ in C++14

Module M53

Partha Pratim
Das

Objectives &
Outlines

λ in C++:
Recap

std::function
Examples

Generic λ

Recursive λ in
C++

Practice Examples

Generic Recursive λ

Practice Examples

Generalized λ
Captures

Module Summary

```
// Sorting integers, floats, strings using a generalized lambda and sort function
// #include <iostream>, #include <string>, #include <vector>, #include <algorithm>, using namespace std;

// Utility Function to print the elements of a collection
void printElements(auto& C) {
    for (auto e : C) cout << e << " ";
    cout << endl;
}

int main() {
    // Declare a generalized lambda and store it in greater. Works for int, double, string, ...
    auto greater = [](auto a, auto b) -> bool { return a > b; };

    vector<int> vi = { 1, 4, 2, 1, 6, 62, 636 }; // Initialize a vector of integers
    vector<double> vd = { 4.62, 161.3, 62.26, 13.4, 235.5 }; // Initialize a vector of doubles
    vector<string> vs = { "Tom", "Harry", "Ram", "Shyam" }; // Initialize a vector of strings

    sort(vi.begin(), vi.end(), greater); // Sort integers
    sort(vd.begin(), vd.end(), greater); // Sort doubles
    sort(vs.begin(), vs.end(), greater); // Sort strings

    printElements(vi); // 636 62 6 4 2 1 1
    printElements(vd); // 235.5 161.3 62.26 13.4 4.62
    printElements(vs); // Tom Shyam Ram Harry
}
```



Capture-less Generic λ to Function Pointers

Module M53

Partha Pratim Das

Objectives & Outlines

λ in C++:
Recap

std::function

Examples

Generic λ

Recursive λ in C++

Practice Examples

Generic Recursive λ

Practice Examples

Generalized λ Captures

Module Summary

- A non-generic λ with an empty capture-list can be converted to a function pointer
`typedef int (*fp) (int); // Function pointer`
`const fp f = [](int i){ return i; }; // Converts to a func. ptr. w/o capture`

- A capture-less generic λ behaves in the same way
- Further, it *can be converted to more than one compatible function pointers*

```
#include <iostream>
```

```
void f(void(*fp)(int)) { fp(1); /*...*/ }
```

```
void g(void(*fp)(double)) { fp(2.2); /*...*/ }
```

```
int main () {  
    auto op = [](auto x) { // generic code for x  
        std::cout << "x = " << x << std::endl;  
    };
```

```
    // use 'op' as a generic callback function pointer
```

```
    f(op); // x = 1
```

```
    g(op); // x = 2.2
```

```
}
```




Recursive λ in C++

Module M53

Partha Pratim
Das

Objectives &
Outlines

λ in C++:
Recap

`std::function`

Examples

Generic λ

Recursive λ in
C++

Practice Examples

Generic Recursive λ

Practice Examples

Generalized λ
Captures

Module Summary

Source:

- [Recursive lambda expressions in C++](#), [geeksforgeeks.org](#)
- [Recursive lambda functions in C++11](#), [stackoverflow.com](#)
- [std::function](#), [cppreference.com](#)
- [Usage and Syntax of std::function](#), [stackoverflow.com](#)
- [std::function](#), [cplusplus.com](#)
- [std::function::function](#), [cplusplus.com](#)

Recursive λ in C++



Recursive λ in C++11

Module M53

Partha Pratim Das

Objectives & Outlines

λ in C++:
Recap

std::function

Examples

Generic λ

Recursive λ in C++

Practice Examples

Generic Recursive λ

Practice Examples

Generalized λ
Captures

Module Summary

- A recursive function *CountOnes*(*n*) counts the number of 1's in the binary of *n*:

$$\begin{aligned}
 \text{CountOnes}(n) &= \text{CountOnes}((n-1)/2) + 1, n > 0 \ \& \ \text{odd} \\
 &= \text{CountOnes}(n/2), n > 0 \ \& \ \text{even} \\
 &= 0, n = 0
 \end{aligned}$$

For example, *CountOnes*(5) = 2 (5 \equiv 101) and *CountOnes*(14) = 3 (14 \equiv 1110)

- Coded as a normal recursive function:

```

#include <iostream>      // std::cout

// recursive function CountOnes
int CountOnes(int n) { return (0 == n)? 0: CountOnes(n/2) + (n % 2); }

int main() {
    auto Print = // no capture needed for CountOnes or std::cout
                // as they are available as global symbols
    [](int n) { std::cout << "#" << n << " = " << CountOnes(n) << "; "; };

    Print(0); Print(1); Print(2); Print(3); Print(1729);
    // #(0) = 0; #(1) = 1; #(2) = 1; #(3) = 2; #(1729) = 5;
}

```



Recursive λ in C++11

Module M53

Partha Pratim Das

Objectives & Outlines

λ in C++:
Recap

std::function

Examples

Generic λ

Recursive λ in C++

Practice Examples

Generic Recursive λ

Practice Examples

Generalized λ
Captures

Module Summary

- An attempt to code `CountOnes` as a lambda runs into compilation error:

```
#include <iostream>      // std::cout

int main() {
    auto CountOnes = [] (int n) { // error: use of 'CountOnes' before deduction of 'auto'
        return (0 == n)? 0: CountOnes(n/2) + (n % 2);
    };
    // ...
}
```

- `std::function` allows to declare the signature of `CountOnes` before defining it as a λ :

```
#include <iostream>      // std::cout
#include <functional>    // std::function

int main() {
    std::function<int(int)> CountOnes; // signature of CountOnes
    CountOnes = // capture CountOnes as its use is free in the body
        [&CountOnes] (int n) -> int { return (0 == n)? 0: CountOnes(n/2) + (n % 2); };

    auto Print = // capture needed for CountOnes now as it is a local symbol
        [&CountOnes](int n) { std::cout << "(" << n << ") = " << CountOnes(n) << "; "; };

    Print(0); Print(1); Print(2); Print(3); Print(1729);
    // #(0) = 0; #(1) = 1; #(2) = 1; #(3) = 2; #(1729) = 5;
}
```



Example 1: Co-recursive Functions

Module M53

Partha Pratim
Das

Objectives &
Outlines

λ in C++:
Recap

`std::function`

Examples

Generic λ

Recursive λ in
C++

Practice Examples

Generic Recursive λ

Practice Examples

Generalized λ
Captures

Module Summary

```
#include <iostream>
#include <functional>

int main() {
    std::function<int(int)> f1;
    std::function<int(int)> f2 =
        [&](int i) {
            std::cout << i << " ";
            if (i > 5) { return f1(i - 2); } else { return 0; }
        };

    f1 = [&](int i) { std::cout << i << " "; return f2(++i); };

    f1(10);

    return 0;
}
```

10 11 9 10 8 9 7 8 6 7 5 6 4 5



Example 2: Factorial

Module M53

Partha Pratim
Das

Objectives &
Outlines

λ in C++:
Recap

std::function

Examples

Generic λ

Recursive λ in
C++

Practice Examples

Generic Recursive λ

Practice Examples

Generalized λ
Captures

Module Summary

```
#include <iostream>
#include <functional>

int main() {
    std::function<int(int)> fact;
    fact =
        [&fact](int n) -> int
        { return (n == 0) ? 1 : (n * fact(n - 1)); };

    std::cout << "factorial(4) : " << fact(4) << std::endl;

    return 0;
}
```



Example 3: Fibonacci

Module M53

Partha Pratim
Das

Objectives &
Outlines

λ in C++:
Recap

std::function

Examples

Generic λ

Recursive λ in
C++

Practice Examples

Generic Recursive λ

Practice Examples

Generalized λ
Captures

Module Summary

```
#include <iostream>
#include <functional>
using namespace std;

int main() {
    std::function<int(int)> fibo;
    fibo =
        [&fibo](int n)->int
        { return (n == 0) ? 0 :
                  (n == 1) ? 1 :
                  (fibo(n - 1) + fibo(n - 2)); };

    cout << "fibo(8) : " << fibo(8) << endl;

    return 0;
}
```



Generic Recursive λ in C++14

Module M53

Partha Pratim Das

Objectives & Outlines

λ in C++:
Recap

`std::function`

Examples

Generic λ

Recursive λ in C++

Practice Examples

Generic Recursive λ

Practice Examples

Generalized λ
Captures

Module Summary

- A λ does not have a named specific type. So a recursive λ expression needs `std::function` wrapper
- The generic λ expression in C++14 allows recursive λ functions without using `std::function`
- Consider the `CountOnes` example again:

```
#include <iostream>      // std::cout

int main() {
    auto CountOnes = // we need to pass CountOnes as a parameter in the lambda
                    // note the use of CountOnes as the first parameter in the call
    [] (auto&& CountOnes, int n) -> int { // C++14
        return (0 == n)? 0: CountOnes(CountOnes, n/2) + (n % 2);
    };

    auto Print = // capture needed for CountOnes now as it is a local symbol
    [&CountOnes] (int n)
    { std::cout << "CountOnes(" << n << ") = " << CountOnes(CountOnes, n) << std::endl; };

    Print(0); Print(1); Print(2); Print(3); Print(1729);
    // #(0) = 0; #(1) = 1; #(2) = 1; #(3) = 2; #(1729) = 5;
}
```



Generic Recursive λ in C++14: Power

Module M53

Partha Pratim Das

Objectives & Outlines

λ in C++:
Recap

`std::function`

Examples

Generic λ

Recursive λ in C++

Practice Examples

Generic Recursive λ

Practice Examples

Generalized λ
Captures

Module Summary

```
#include <iostream> // C++11 solution using std::function. Works for int base
#include <functional>
int main() {
    std::function<int(int,int)> pow; // pow recursive function. std::function used to define type of pow
    pow = [&pow](int base, int exp) { return exp==0 ? 1 : base*pow(base, exp-1); };
    std::cout << pow(2, 10); // 2^10 = 1024
    std::cout << pow(2.71828, 10); // e^10 = 1024 // 2.71828 is cast to int giving 2 and wrong result
}

#include <iostream> // C++14 solution without using std::function. Works for any numeric base
int main() {
    auto power = [](auto self, auto base, int exp) -> decltype(base) { // any numeric 'base' type
        return exp==0 ? 1 : base*self(self, base, exp-1);
    };
    // Wrapper of power to avoid passing power as first parameter to the call
    auto pow = [power](auto base, int exp) -> decltype(base) { return power(power, base, exp); };

    std::cout << power(power, 2, 10); // 2^10 = 1024 // Needs to pass itself as first parameter
    std::cout << power(power, 2.71828, 10); // e^10 = 22026.3

    std::cout << pow(2, 10); // 2^10 = 1024 // Wrapper provides a clean solution
    std::cout << pow(2.71828, 10); // e^10 = 22026.3
}
```




Generic Recursive λ in C++14: Factorial

Module M53

Partha Pratim Das

Objectives & Outlines

λ in C++:
Recap

std::function
Examples

Generic λ

Recursive λ in C++

Practice Examples
Generic Recursive λ
Practice Examples

Generalized λ
Captures

Module Summary

```
#include <iostream> // Factorial lambda in C++11 solution using std::function
#include <functional>
int main () {
    std::function < int (int) > fact;
    fact = [&fact](int n) -> int
        { return (n == 0) ? 1 : (n * fact(n - 1));
    };
    std::cout << "factorial(4) : " << fact(4) << std::endl;
}
```

```
#include <iostream> // Factorial lambda in C++14 without using std::function
int main () {
    auto factorial = [](auto self, int n) -> int
        { return (n == 0) ? 1 : (n * self(self, n - 1));
    };
    auto fact = [factorial](int n) -> int // Wrapper of factorial to skip passing factorial
        { return factorial(factorial, n);
    };
    std::cout << "factorial(4) : " << fact(4) << std::endl;
}
```



Generic Recursive λ in C++14: Fibonacci

Module M53

Partha Pratim
Das

Objectives &
Outlines

λ in C++:
Recap

std::function

Examples

Generic λ

Recursive λ in
C++

Practice Examples

Generic Recursive λ

Practice Examples

Generalized λ
Captures

Module Summary

```
#include <iostream> // Fibonacci lambda in C++11 solution using std::function
#include <iostream>
#include <functional>
int main () {
    std::function < int (int) > fibo;
    fibo = [&fibo](int n) -> int {
        return (n == 0) ? 0 : (n == 1) ? 1 : (fibo(n - 1) + fibo(n - 2));
    };
    std::cout << "fibo(8) : " << fibo(8) << std::endl;
}
```

```
#include <iostream> // Fibonacci lambda in C++14 without using std::function
int main () {
    auto fibonacci = [](auto self, int n) -> int {
        return (n == 0) ? 0 : (n == 1) ? 1 : (self(self, n - 1) + self(self, n - 2));
    };
    auto fibo = [fibonacci](int n) -> int { // Wrapper of fibonacci to skip passing fibonacci
        return fibonacci(fibonacci, n);
    };
    std::cout << "fibo(8) : " << fibo(8) << std::endl;
}
```



Generalized λ Captures

Module M53

Partha Pratim Das

Objectives & Outlines

λ in C++:
Recap

`std::function`

Examples

Generic λ

Recursive λ in C++

Practice Examples

Generic Recursive λ

Practice Examples

Generalized λ
Captures

Module Summary

Generalized λ Captures

Source:

- [Generalized lambda captures](https://ericniebler.com/2015/07/lambda-captures/), isocpp.org



Generalized λ Captures in C++14

Module M53

Partha Pratim Das

Objectives & Outlines

λ in C++:
Recap

`std::function`
Examples

Generic λ

Recursive λ in C++

Practice Examples
Generic Recursive λ
Practice Examples

Generalized λ Captures

Module Summary

- In C++11, λ s could not (easily) capture by move. Generalized λ capture in C++14 solves the problem and also allows to define arbitrary new local variables in the λ object:

```
auto u = make_unique<some_type>(some, parameters); // a unique_ptr is move-only (TBD later)
go.run([u=move(u)] { do_something_with(u); }); // move the unique_ptr into the lambda
```

- We can also rename the variable `u` the same inside the λ

```
go.run([u2=move(u)] { do_something_with(u2); }); // capture as u2
```

- And we can add arbitrary new state to the λ object, because each capture creates a new type-deduced local variable inside the λ :

```
int x = 4;
int z = [&r = x, y = x + 1] { // y is local, set to 5
    r += 2;                  // set x to 6; "r is for Renamed Ref"
    return y + 2;            // return 7 to initialize z
};                           // invoke lambda
```



Module Summary

Module M53

Partha Pratim
Das

Objectives &
Outlines

λ in C++:
Recap

`std::function`

Examples

Generic λ

Recursive λ in
C++

Practice Examples

Generic Recursive λ

Practice Examples

Generalized λ
Captures

Module Summary

- Learnt different techniques without or with `std::function` to write and use non-recursive and recursive λ expressions in C++11 / C++14
- Several practice examples to be tried and tested