# Lambda expression in C++

C++ 11 introduced lambda expression to allow us write an inline function which can be used for short snippets of code that are not going to be reuse and not worth naming. In its simplest form lambda expression can be defined as follows:

```
[ capture clause ] (parameters) -> return-type
{
   definition of method
}
```

Generally return-type in lambda expression are evaluated by compiler itself and we don’t need to specify that explicitly and -> return-type part can be ignored but in some complex case as in conditional statement, compiler can’t make out the return type and we need to specify that.
Various uses of lambda expression with standard function are given below :

```
// C++ program to demonstrate lambda expression in C++
#include <bits/stdc++.h>
using namespace std;

// Function to print vector
void printVector(vector<int> v)
{
    // lambda expression to print vector
    for_each(v.begin(), v.end(), [](int i)
    {
        std::cout << i << " ";
    });
```

```
        cout << endl;
    }

    int main()
    {
        vector<int> v {4, 1, 3, 5, 2, 3, 1, 7};

        printVector(v);

        // below snippet find first number greater than 4
        // find_if searches for an element for which
        // function(third argument) returns true
        vector<int>:: iterator p = find_if(v.begin(), v.end(), [](int
        {
            return i > 4;
        });
        cout << "First number greater than 4 is : " << *p << endl;


        // function to sort vector, lambda expression is for sorting
        // non-decreasing order Compiler can make out return type as
        // bool, but shown here just for explanation
        sort(v.begin(), v.end(), [](const int& a, const int& b) -> bo
        {
            return a > b;
        });

        printVector(v);

        // function to count numbers greater than or equal to 5
        int count_5 = count_if(v.begin(), v.end(), [](int a)
        {
            return (a >= 5);
        });
        cout << "The number of elements greater than or equal to 5 is
             << count_5 << endl;

        // function for removing duplicate element (after sorting all
        // duplicate comes together)
        p = unique(v.begin(), v.end(), [](int a, int b)
        {
            return a == b;
        });

        // resizing vector to make size equal to total different numb
        v.resize(distance(v.begin(), p));
        printVector(v);

        // accumulate function accumulate the container on the basis
        // function provided as third argument
        int arr[] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
        int f = accumulate(arr, arr + 10, 1, [](int i, int j)
        {
            return i * j;
    });

    cout << "Factorial of 10 is : " << f << endl;

    //      We can also access function by storing this into varia
    auto square = [](int i)
    {
        return i * i;
    };

    cout << "Square of 5 is : " << square(5) << endl;
}
```

Output:

```
4 1 3 5 2 3 1 7
First number greater than 4 is : 5
7 5 4 3 3 2 1 1
The number of elements greater than or equal to 5 is : 2
7 5 4 3 2 1
Factorial of 10 is : 3628800
Square of 5 is : 25
```

A lambda expression can have more power than an ordinary function by having access to variables from the enclosing scope. We can capture external variables from enclosing scope by three ways :

Capture by reference
Capture by value
Capture by both (mixed capture)

Syntax used for capturing variables :

[&] : capture all external variable by reference
[=] : capture all external variable by value
[a, &b] : capture a by value and b by reference

A lambda with empty capture clause [ ] can access only those variable which are local to it.
Capturing ways are demonstrated below :

```
// C++ program to demonstrate lambda expression in C++
#include <bits/stdc++.h>
using namespace std;

int main()
{
    vector<int> v1 = {3, 1, 7, 9};
    vector<int> v2 = {10, 2, 7, 16, 9};

    //  access v1 and v2 by reference
    auto pushinto = [&] (int m)
    {
        v1.push_back(m);
        v2.push_back(m);
    };

    // it pushes 20 in both v1 and v2
    pushinto(20);

    // access v1 by copy
    [v1]()
    {
        for (auto p = v1.begin(); p != v1.end(); p++)
        {
            cout << *p << " ";
        }
    };

    int N = 5;

    // below snippet find first number greater than N
    // [N]  denotes,   can access only N by value
    vector<int>:: iterator p = find_if(v1.begin(), v1.end(), [N](
    {
        return i > N;
    });

    cout << "First number greater than 5 is : " << *p << endl;

    // function to count numbers greater than or equal to N
    // [=] denotes,   can access all variable
    int count_N = count_if(v1.begin(), v1.end(), [=](int a)
    {
        return (a >= N);
    });

    cout << "The number of elements greater than or equal to 5 is
         << count_N << endl;
}
```

Output:

```
First number greater than 5 is : 7
The number of elements greater than or equal to 5 is : 3
```

Lambda expression can work only on C++ 11 and after versions.

This article is contributed by Utkarsh Trivedi. Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above

## Recommended Posts:

Generalized Lambda Expressions in C++14

Can we use function on left side of an expression in C and C++?

Increment (Decrement) operators require L-value Expression

std::regex_match, std::regex_replace() | Regex (Regular Expression) In C++

fill in C++ STL

Conditional or Ternary Operator (?:) in C/C++

forward_list insert_after() function in C++ STL

Count of distinct remainders when N is divided by all the numbers from the range [1, N]

C++ | asm declaration

Pointers and References in C++

Strings in C++ and How to Create them?

Enum Classes in C++ and Their Advantage over Enum DataType

Deque vs Vector in C++ STL

C++ Programming Basics

**Article Tags :** C C++ cpp-advanced

Practice Tags : C CPP

9