LAMBDA EXPRESSION X

.:: Fundamentals ::.

- 1. <u>CLÁUSULA DE CAPTURA</u> (también conocida como *iniciador de expresión lambda* en la especificación de C++)
- 2. <u>LISTA DE PARÁMETROS</u> Opcional (también conocida como *declarador de expresión lambda*)
- 3. **ESPECIFICACIÓN MUTABLE** Opcional
- 4. <u>ESPECIFICACIÓN DE EXCEPCIÓN</u> Opcional
- 5. <u>TIPO DE VALOR DEVUELTO FINAL</u> Opcional
- 6. CUERPO DE LA EXPRESIÓN lambda



```
/* 1 . Declarar expresiones lambda */-
 1
 2
 3
     #include <functional>
     #include <iostream>
 4
 5
 6
     using namespace std;
 7
 8
    int main(){-
 9
    auto f1 = [](int x, int y) { return x + y; };
10
11
    cout << f1(2, 3) << endl;-</pre>
12
    ---//-
    function(int, int)> f2 = [](int x, int y) { return x + y; };
13
14
     cout << f2(3, 4) << endl;</pre>
15
    ---//-
16
        auto cuadrado = [](int x) { return x * x; }(5);
    cout << cuadrado << endl;</pre>
17
18
19
20
    }-
21
```



```
/* 2. Llamar a una expresion lambda */-
1
2
    #include <iostream>
3
     using namespace std;
4
    int main(){
5
6
7
     int sum = [] (int x, int y) { return x + y; }(5, 4);
     - · · auto · mul · = · [] · (int · x, · int · y) · { · return · x · * · y; · };
8
9
       cout << "sum(5,4)=> "<< sum << endl;</pre>
10
        cout << "mul(5,4)=> "<< mul(5,3) << endl;
11
     }[
12
```



```
/* 3. Anida expresiones lambda */
 2
     #include <iostream>
 3
 4
 5
     using namespace std;
 6
 7
     int main(){
 8
         int duplicarDosVeces =
 9
10
             [](int x) {
11
                 return [](int y) { return y * 2;}(x)
12
13
             }(5);
14
15
         cout << "duplicar dos veces (5) => "<< duplicarDosVeces << endl;</pre>
16
17
18
    /* 3. Anida expresiones lambda */
1
2
     #include <iostream>
 3
 4
 5
     using namespace std;
 6
7
     int main(){
 8
         int duplicarDosVeces =
 9
             [](int x) {
10
                 return ( [](int y) { return y * 2;}(x)
11
12
                           * 2
13
             }(5);
14
15
16
         cout << "duplicar dos veces (5) => "<< duplicarDosVeces << endl;</pre>
17
18
```



```
// 4. Funciones Lambda de orden superior
   // compile with: /EHsc /W4
3
4
   #include <iostream>
5
   #include <functional>
6
7
   using namespace std;
8
9
   int main(){
10
11
   auto addtwointegers =
12
   13 (int x)
   14
15
   17
   18 ....//----
   auto higherorder =
19
   [ ]....
20
   (const function<int(int)>& f, int z)
21
   { return f(z) * 2; };
22
23
24
   . . . . / / _ _ _ _ . .
25
   auto answer = higherorder(addtwointegers(7), 8);
   cout << answer << endl;</pre>
26
   }¶
27
```

```
// 5.0 Using a Lambda Expression in a Function
2
3
    #include <algorithm>
4
    #include <iostream>
5
    #include <vector>
6
7
    using namespace std;
8
9
    class Scale{
10
    private:
11
       int scale;
12
    public:
13
       explicit Scale(int scale) : scale(scale) {}
14
15
    void ApplyScale(const vector<int>& v) const {
    for each(
16
17
              v.begin(),
18
    v.end(),
    19
    );
20
21
    · · · · }-
    };
22
23
    ///----
24
    int main(){
25
       vector<int> values;
       values.push back(1);
26
       values.push back(2);
27
28
       values.push back(3);
       values.push back(4);
29
30
       Scale s(3);
31
32
       s.ApplyScale(values);
33
34
    }
```

```
// 6.0 template_lambda_expression.cpp
#include <vector>
#include <algorithm>
#include <iostream>
using namespace std;
template <typename T>
void negate_all(vector<T>& v){
    for_each(-
        v.begin(),
        v.end(),
        [](T& n) { n = -n; }
 );
}-
template <typename T>
void print_all(const vector<T>& v){
    for_each(
        v.begin(),
        v.end(),
        [](const T& n) { cout << n << endl;}
. . . . );
}-
int main(){
    vector<int> v;
    v.push_back(34);
    v.push_back(-43);
    v.push_back(56);
    print all(v);
    negate_all(v);
    cout << "After negate_all():" << endl;</pre>
    print all(v);
```

1

2

4

5

6 7

8 9 10

11 12

13

14

15

16

17

18 19

20 21

22

23

24

25

26

27 28 29

30 31

32

33

34

35 36

37

38 39

40