Lambda

09 June 2021 07:07

1. Lambda

Anonymous functor (Functors are objects that contain an overloaded operator() that make them callable like a function) inside another function.

- When the compiler encounters a lambda definition, it creates a custom object definition for the lambda -> Every lambda is given new name
- Each captured variable becomes a data member of the object.
- Parameters become parameters of operator()
- The lambda object is instantiated, and the captured members of the lambda are initialized at that point in constructor of that object.

```
[ captureClause ] ( parameters ) -> returnType
{
    statements;
}
```

```
#include <functional>

int main()
{
  bool check = false;
  std::function<bool(int)> a
  {
    [check](int a) -> bool
    {
      return (check && (a==0));
    }
  };

  a(1);
}
```

```
4  int main()
5  {
6    bool check = false;
7
8    class __lambda_9_5
9    {
10     public:
11     inline /*constexpr */ bool operator()(int a) const
12     {
13       return (check && (a == 0));
14     }
15
16     private:
17     bool check;
18     public:
19     // inline /*constexpr */ __lambda_9_5(const __lambda_9_5 &) noexcept = default;
20     // inline /*constexpr */ __lambda_9_5(__lambda_9_5 &&) noexcept = default;
21     __lambda_9_5(bool & _check)
22     : check{_check}
23     {}
24
25   };
26
27   std::function<bool (int)> a = std::function<bool (int)>{__lambda_9_5{check}};
28   a.operator()(1);
29 }
```

The capture clause and parameters **can both be empty**
The return type **is optional**, and if omitted, **auto will be assumed ->** all return statements in the lambda must return the same type

```
[](){}; // defines a lambda with no captures, no parameters, and no return type
```

2. Lambda syntax

```
auto isEven
{
  [](int i) -> bool
  {
    return ((i % 2) == 0);
  }
};
isEven(int);
```

Auto -> When we write a lambda, the compiler generates a unique type just for the lambda that is not exposed to us.

3. Using function pointer -> cannot be used with captures?

```
bool (*isEven)(int)
{
  [](int i) -> bool
  {
    return ((i % 2) == 0);
  }
};
isEven(int);
```

4. Using std::function

```
std::function<bool(int)> isEven
{
  [](int i) -> bool
  {
    return ((i % 2) == 0);
  }
};
isEven(int);
```

5. Generic Lambda -> Lambda with one or more auto parameter
   compiler will deduce the type of input parameter from the calls made to the lambda
   -> **each type will resolve to a separate lambda OR template specialization of operator()**

   This is equivalent to function template for lambda

6. Lambda in general can access globals, compile-time consts, static duration variables without explicit capture?

7. Lambda capture
   - capture clause is used to (indirectly) give a lambda access to variables available in the surrounding scope that it normally would not have access to.
   - CONST Clone of that variable is made (with an identical name and value) inside the lambda if captured by VALUE
     - Value at the time of constructor call of functor object is captured i.e. at the time of lambda definition is copied.
     - To capture all used variables by value, use a capture value of = used i.e. default arg is always first arg
   - Reference of outside variable is stored in functor object if captured by REFERENCE
     - To capture all used variables by reference, use a capture value of & -> default capture is always first arg for eg [=, &a]
     - Ensure references have scope during lambda call too.. Else dangling reference
   - These cloned variables are initialized with the value of outer scope variables of the same name at this point.
   - Regarding type qualifier of captured variable - **it's a const OR operator() will be const**.. Type may be same
   - Use mutable after parameter list to **remove constness of captured variable OR remove const from operator()**
   - Multiple variables can be captured using comma

```
// Capture health and armor by value, and enemies by reference.
[health, armor, &enemies](){};
// Capture enemies by reference and everything else by value.& is needed
[=, &enemies](){};
// Capture armor by value and everything else by reference.
[&, armor](){};
```

8. Lambda capture contd.
   1. New variable can be introduced in lambda scope without specifying type ---> will be new member variable in functor object

```
[userArea{ 0 }](int knownArea)
{
  return (userArea == knownArea);
}
```

   2. d

I
9. F
10. F

I
11. F

I
12. F

I
13. F

I
14. F

I
15. F

I
16. D

I
17. F

I
18. F

I
19. F

I
20. F
21. F

I
22. F

I
23. F

I
24. F

I

25. F
    I
26. F
    I
27. D
    I
28. F
    I
29. F
    I
30. F
    I
31. F
32. F
    I
33. F
    I
34. F
    I
35. F
    I
36. F
    I
37. F
    I
38. D
    I
39. F
    I
40. F
    I
41. F
    I
42. F
43. F
    I
44. F
    I
45. F
    I
46. F
    I
47. F
    I