Funciones Lambda

Antonio Espín Herranz

Funciones Lambda

- Desde C ++ 11, una función lambda, a menudo denominado un *lambda*: es una forma cómoda de definir un objeto de función anónima (un *cierre - closure*) en la ubicación donde se invoca o se pasa como argumento para una función.
  - También se pueden definir con un nombre.
  - En C++17 se puede utilizar auto en el tipo devuelto y en los parámetros recibidos.
  - Las funciones lambda se crean en tiempo de ejecución.

- Normalmente, las lambdas se usan para encapsular unas líneas de código que se pasan a algoritmos o métodos asincrónicos.

- Las funciones Lambda se pueden almacenar dentro de colecciones tipo vector o map.

Ejemplo

- Se define una función lambda en la llamada a la función sort que ordena un array de float:

```
#include <algorithm>
#include <cmath>

void abssort(float* x, unsigned n) {
    std::sort(x, x + n,
        // Inicio de la función
        [](float a, float b) {
        return (std::abs(a) < std::abs(b));
        } // fin de la lambda
    );
}
```

Partes de la función Lambda

- Hay 3 partes en una lambda:
  - Lista de captura, representada por unos: []
  - Lista de parámetros (opcional). Representada por unos: ()
  - Cuerpo de la función. Representada por: {}

- Las 3 partes pueden estar vacías:
- []{}
  - Lambda vacía, que no recibe parámetros, no devuelve nada.

Lista de Captura []

- Por defecto, **no se puede acceder a las variables del ámbito adjunto** por un lambda.

- Capturar una variable lo hace accesible dentro de la lambda, ya sea como una **copia** o como una **referencia** .

- Las variables capturadas se convierten en parte de la lambda; en contraste con los argumentos de la función, no se tienen que pasar al llamar a lambda.

Ejemplos

```
// Define una variable int, fuera de la lambda
int a = 0;

// Error: 'a' no es accesible
auto f = []()   { return a*9; };

// OK, se captura 'a' por valor
auto f = [a]()  { return a*9; };

// OK, 'a' capturada por referencia
auto f = [&a]() { return a++; };

// Llamada a la Lambda, sin parámetros.
auto b = f();
```

Ejemplos

- **[a, &b] () {...}**
  - Se captura a por copia y b por referencia.

- **[&, a] () {...}**
  - Se captura una copia (la a) y cualquier otra variable utilizada por referencia.

- **[=, &b, i{22}, this] () {…}**
  - Esto captura b por referencia, this por copia, inicializa una nueva variable i con valor 22 y captura cualquier otra utilizada variable por copia.

std::function

- Normalmente utilizaremos auto para recoger la función lambda que estamos definiendo en un momento dado.

- Disponemos del objeto:
  - **std::function** definido dentro del fichero de cabecera: **<functional>**

- Ejemplo:

```
std::function f = [](int a, int b){ return a+b+; };
std::cout << “Resultado: “ << f(8,9) << std::endl;
```

Lista de Parámetros ()

- Si la lambda no toma argumentos, estos paréntesis se pueden omitir (excepto si necesita declarar la lambda **mutable**).

- Estas dos lambdas son equivalentes: (con paréntesis y sin paréntesis)
  - auto call_foo = [x]**()**{ x.foo(); };
  - auto call_foo2 = [x]{ x.foo(); };

Lista de Parámetros ()

- En C ++ 14 y 17
- La lista de parámetros puede usar el tipo de marcador de posición **auto** en lugar de los tipos reales.
- Al hacerlo, este argumento se comporta como un parámetro de plantilla de una plantilla de función.
- Las siguientes lambdas son equivalentes cuando desea ordenar un vector en código genérico:

Ejemplo

```
auto sort_cpp11 = []
(
    std::vector<T>::const_reference lhs,
    std::vector<T>::const_reference rhs
) { return lhs < rhs; };

auto sort_cpp14 = []
(
    const auto &lhs,
    const auto &rhs
) { return lhs < rhs; };
```

Cuerpo de la Función {}

- El objeto de resultado de una expresión lambda es un cierre , que se puede llamar usando el **operator()** (como con otros objetos de función):

```
int multiplier = 5; // Captura por valor
auto timesFive = [multiplier](int a) { return a * multiplier; };
std::cout << timesFive(2); // Imprime 10
multiplier = 15;
std::cout << timesFive(2); // Todavía imprime 2*5 == 10
```

Tipo de Retorno

- De forma predeterminada, se deduce el tipo de retorno de una expresión lambda.

- En este ejemplo, retorna un bool:

```
[](){ return true; };
```

- También puede especificar manualmente el tipo de retorno usando la siguiente sintaxis:

```
[]() -> bool { return true; };
```

Lambda mutable

- Los objetos capturados por valor en la lambda son, por defecto, inmutables.
- Esto se debe a que el operator**()** del objeto de cierre generado es **const** de forma predeterminada.

- // Falla al compilar porque ++c intenta modificar el estado de la lambda.
- auto func = [c = 0](){++c; std::cout << c;};

- Se puede utilizar el modificador **mutable**.
- auto func = [c = 0]() **mutable** {++c; std::cout << c;};

- Si se indicara el tipo de retorno, quedaría así: No se suele indicar el tipo de retorno.
- auto func = [c = 0]() **mutable -> int** {++c; std::cout << c; return c;};

Ejemplo: uso de Lambdas

- std::vector<int> vec{ 1, 2, 3, 4, 5 };

- // Encuentra un número que sea menor que una entrada dada.
- int = 10;
- auto it = std::find_if(vec.begin(), vec.end(), [threshold](int value) { return value < threshold; });