

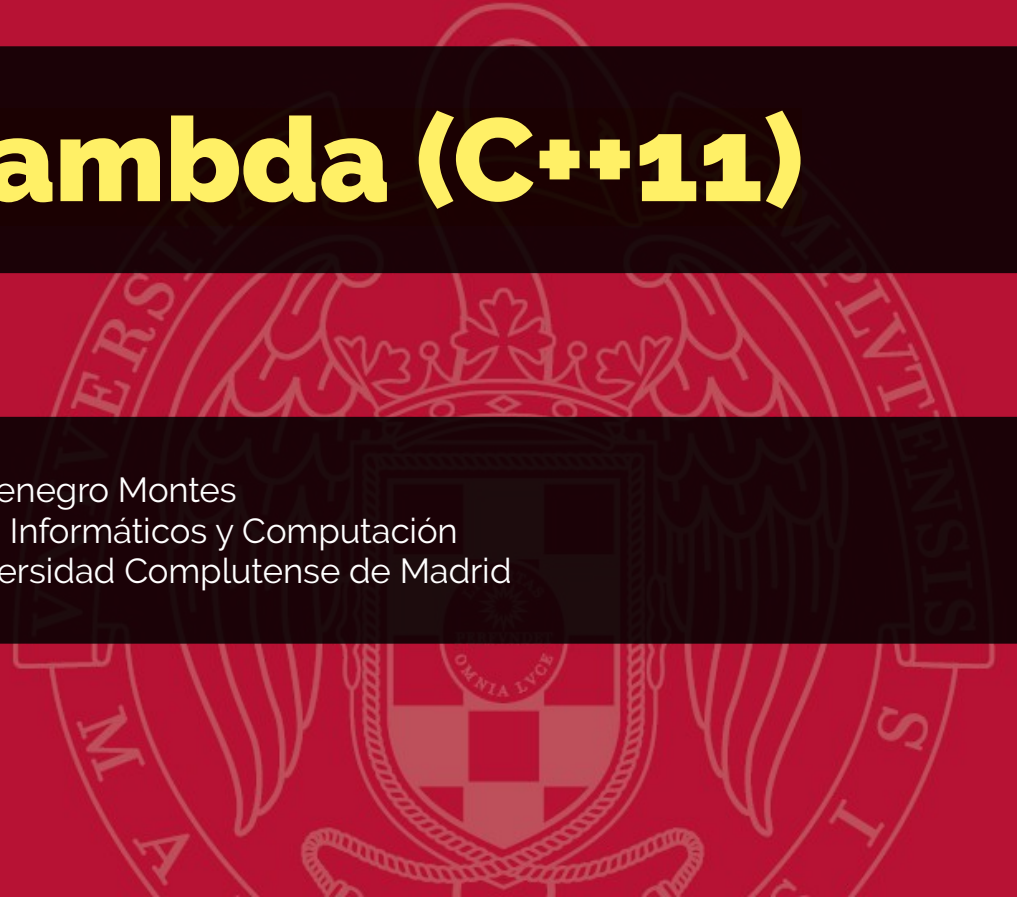
ESTRUCTURAS DE DATOS

NOTAS SOBRE C++

# Expresiones lambda (C++11)

Manuel Montenegro Montes

Departamento de Sistemas Informáticos y Computación  
Facultad de Informática – Universidad Complutense de Madrid



# Recordatorio

Solución para no tener que escribir tanto código para los objetos función

- Función que recibe una lista, una función booleana y elimina aquellos elementos para los que la función devuelve `true`.

```
template <typename T, typename U>
void eliminar(std::list<U> &elems, T func) {
    auto it = elems.begin();
    while (it != elems.end()) {
        if (func(*it)) {
            it = elems.erase(it);
        } else {
            ++it;
        }
    }
}
```

# Recordatorio

Hasta ahora hemos pasado como argumento func:

- **Funciones:**

```
bool es_par(int x) { return x % 2 == 0; }  
...  
eliminar(v1, es_par);
```

Definir aquí mismo nuestra función u objeto función

eliminar (-; ✓)

- **Objetos función:**

```
class EsMultiploDeY { ... }  
...  
EsMultiploDeY multiplo_de_dos(2);  
eliminar(v1, multiplo_de_dos);
```

Esto es demasiado laborioso

- En cualquier caso, tenemos que definir una función o una clase aparte.
  - y es posible que solamente se utilice una vez.

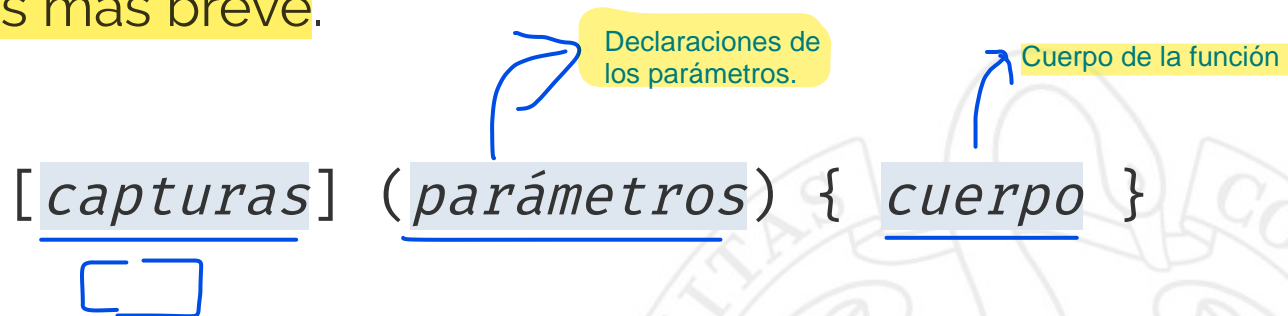
# Expresiones lambda

- Nos permiten declarar un objeto función en el sitio en el que se utiliza, con una sintaxis más breve.
- Sintaxis:

[*capturas*] (*parámetros*) { *cuerpo* }

Declaraciones de los parámetros.

Cuerpo de la función



The diagram illustrates the syntax of a lambda expression: [*capturas*] (*parámetros*) { *cuerpo* }. The *capturas* part is underlined and has a blue bracket below it. The *parámetros* part is underlined and has a blue arrow pointing to it from a yellow box labeled 'Declaraciones de los parámetros.'. The *cuerpo* part is underlined and has a blue arrow pointing to it from a yellow box labeled 'Cuerpo de la función'. The background features a large, faint watermark of the University of Complutense seal.

# Ejemplo

- En lugar de

```
bool es_par(int x) { return x % 2 == 0; }  
...  
eliminar(v1, es_par);
```

- Puede escribirse

```
eliminar(v1, [](int x) { return x % 2 == 0; });
```

Esto es como lo que hemos visto en Java

# Más ejemplos

```
std::list<int> v1 = {1, 5, 6, 9, 10, 20};  
std::list<int> v2 = v1;  
eliminar(v1, [](int x) { return x % 2 == 0; });  
std::cout << v1 << std::endl;
```

dado un número x nos dice si ese número es par.

Impar

```
eliminar(v2, [](int x) { return x % 2 == 1; });  
std::cout << v2 << std::endl;
```

```
std::list<int> v3 = {-2, 3, 10, -6, 20};  
eliminar(v3, [](int x) { return x > 0; });  
std::cout << v3 << std::endl;
```

```
std::list<Fecha> v4 = { {25, 12, 2010}, {10, 21, 2020}, {25, 12, 1900}, {1, 1, 2000} };  
eliminar(v4, [](const Fecha &f) { return f.get_dia() == 25 && f.get_mes() == 12; });  
std::cout << v4 << std::endl;
```

Dado una fecha f devuelva true si es Navidad si no me equivoco

# Capturas

- Las expresiones lambda pueden tener, en su cuerpo, referencias a variables *externas* (esto es, variables distintas a los parámetros).

```
int y = 3;  
eliminar(v, [](int x) { return x % y = 0; });
```

Variable externa.

- Cuando esto ocurre, decimos que la variable *y* está **capturada** por la expresión lambda.
- C++ nos obliga a declarar las variables capturadas dentro de `[]`.

```
int y = 3;  
eliminar(v, [y](int x) { return x % y = 0; });
```

IMPORTANTE. ESTÁ EXPRESIÓN CAPTURA LA Y

# Capturas

Hay dos maneras de capturar variables:

- **Por valor**

```
[y](int x) { /* ... */ }
```

Dentro de la lambda expresión no se pueden realizar cambios sobre la variable y.

- **Por referencia**

Con esto si que podemos modificar la variable y y esto se reflejará en la variable.

```
[&y](int x) { /* ... */ }
```

La lambda expresión trabaja con una **referencia** a la variable y.

Cualquier cambio que se haga sobre la variable y dentro de la lambda expresión afectará a la variable y externa.



# Ejemplo

Entonces la y si se modifica, obtendremos un 4.

```
int y = 3;  
auto f = [&y]() { y++; };  
f();  
std::cout << y << std::endl;
```



# Criba de eratóstenes: el retorno

```
std::list<int> lista;  
std::list<int> primos;  
...  
  
while (!lista.empty()) {  
    int primero = lista.front();  
    primos.push_back(primero);  
    eliminar(lista, [primero](int x) { return x % primero == 0; });  
}  
  
std::cout << primos << std::endl;
```

Elimina aquellos para los que la función devuelve true. primero es externa y la tenemos que capturar.