

ESTRUCTURAS DE DATOS

NOTAS SOBRE C++

Objetos función

Manuel Montenegro Montes

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



Recordatorio

Con esto nos va a explotar la cabeza según él en el vídeo.

- 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;
        }
    }
}
```

Como esta función eliminar recibe como parámetro una función se trata de una FUNCIÓN DE ORDEN SUPERIOR.

¿Qué puedo pasar como parámetro func?

```
template <typename T, typename U>
void eliminar(std::list<U> &elems, T func) {
    ...
    if (func(*it)) { ... }
    ...
}
```

— Esta T es el tipo que tiene la función

- Cualquier cosa sobre la que se pueda realizar una llamada.
 - En particular, cualquier función que acepte un solo parámetro.
 - ...¿algo más?

Podemos pasar algo más además de las funciones que se comportan como un parámetro de la función

¿Qué operadores pueden sobrecargarse?

+ - * / % ^ & | << >>

== <= >= != < > && || !

= += -= *= /=

++ --

[] () →

new delete

etc.

 Los que hemos visto hasta ahora

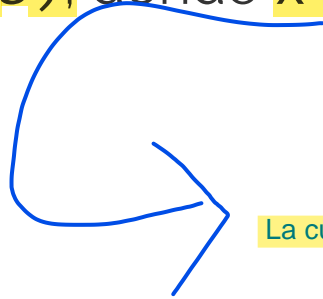
 El que vamos a ver ahora.

Sobrecarga del operador ()

- C++ permite sobrecargar el operador ().

```
class Prueba {  
public:  
    void operator()(parametros) { ... }  
};
```

- Este operador es invocado cuando se evalúa una expresión de la forma `x(args)`, donde `x` es una instancia de la clase `Prueba`.



La cual sobrecarga el operador paréntesis

Ejemplo

```
class SumaUno {  
public:  
    int operator()(int x) { return x + 1; }  
};
```

Al sobrecargarlo recibes un entero x y devuelves ese entero x más uno

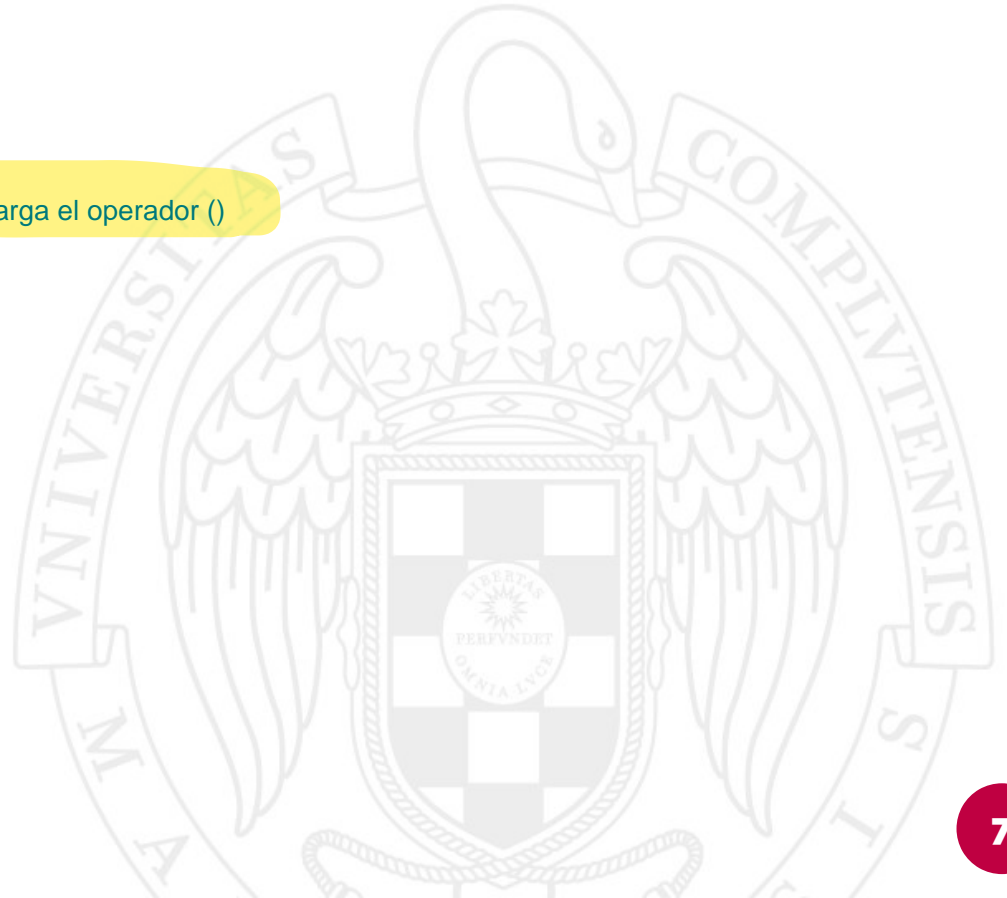
- Supongamos que declaro una instancia de la clase SumaUno:
SumaUno s;
- La expresión s(3) equivale a s.operator()(3) y se evaluará al valor 4.
- ¡Ojo! s no es una función; es un objeto que se comporta como una función.

PERO NO ES UNA FUNCIÓN

Objetos función

- Un **objeto función** es una instancia de una clase que sobrecarga el operador ().
- En nuestro ejemplo:
`SumaUno s;`
s es un objeto función.

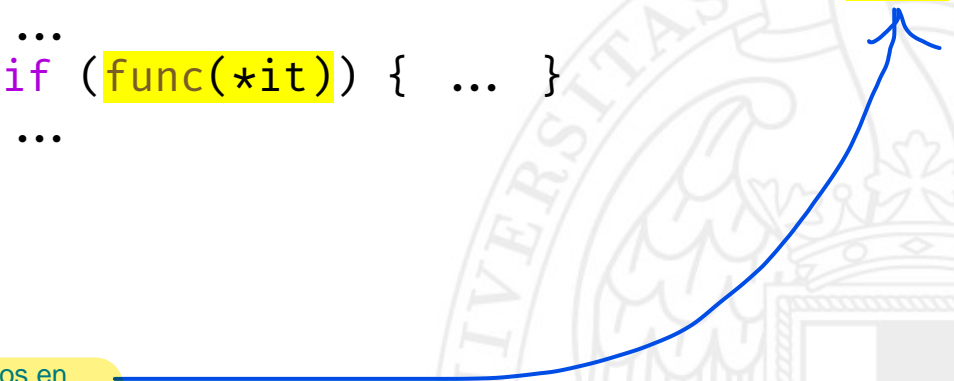
Ya que esta clase sobrecarga el operador ()



Uso de los objetos función

- Los objetos función pueden ser utilizados en cualquier contexto en el que se requiera una función.

```
template <typename T, typename U>
void eliminar(std::list<U> &elems, T func) {
    ...
    if (func(*it)) { ... }
    ...
}
```



Por tanto, como pueden ser utilizados en contextos en los que se requiera una función, aquí podríamos añadir el objeto función

Ejemplo

```
class EsPar {  
public:  
    bool operator()(int x) { return x % 2 == 0; }  
};
```

sobrecarga el operador paréntesis.

```
int main() {  
    ...  
    EsPar obj_fun;  
    eliminar(v1, obj_fun);  
    ...  
}
```

Objeto función

¿Para qué sirven los objetos función?



¿Cuál es la diferencia?

Entre esto...

```
class EsPar {  
public:  
    bool operator()(int x) { return x % 2 == 0; }  
};
```

...y esto...

```
bool es_par(int x) { return x % 2 == 0; }
```

Hay casos en los que se necesitan objetos función

Ejemplo: criba de Eratóstenes

Me quedo con el 2 tachando sus múltiplos.
Me quedo con el siguiente NO tachado,
en este caso el 3, lo cojo y tachando sus
múltiplos, así sucesivamente.

lista de números

	2	3	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

Para obtener número primo, un número primo es un número natural mayor que uno que tiene 2 divisores. Uno y él mismo

Ejemplo: criba de Eratóstenes

- Supongamos que tenemos una lista con los números 2, 3, 4, 5, ..., 100.
 - Eliminamos los múltiplos de 2.
 - Eliminamos los múltiplos de 3.
 - Eliminamos los múltiplos de 5.
 - etc.

Para hacer esto llamamos a la función eliminar que hemos implementado

Ejemplo: criba de Eratóstenes

```
bool es_multiplo_de_dos(int x) {  
    return x % 2 == 0;  
}  
  
bool es_multiplo_de_tres(int x) {  
    return x % 3 == 0;  
}  
  
bool es_multiplo_de_cinco(int x) { ... }  
  
int main() {  
    std::list<int> lista;  
    for (int i = 2; i ≤ 100; i++) { lista.push_back(i); }  
  
    std::list<int> primos;  
    primos.push_back(lista.front());  
    eliminar(lista, es_multiplo_de_dos);  
    primos.push_back(lista.front());  
    eliminar(lista, es_multiplo_de_tres);  
    primos.push_back(lista.front());  
    eliminar(lista, es_multiplo_de_cinco);  
    ...  
}
```

Función para saber si un número es múltiplo de otro.

no parece muy sensato tanta repetición

Lista de números del 2 al 100

Ejemplo: criba de Eratóstenes

```
bool es_multiplo_de_y(int x, int y) {  
    return x % y == 0;  
}
```

```
int main() {  
    std::list<int> lista;  
    for (int i = 2; i ≤ 100; i++) { lista.push_back(i); }  
  
    std::list<int> primos;  
    primos.push_back(lista.front());  
    eliminar(lista, es_multiplo_de_y);  
    primos.push_back(lista.front());  
    eliminar(lista, es_multiplo_de_y);  
    primos.push_back(lista.front());  
    eliminar(lista, es_multiplo_de_y);  
    ...  
}
```

Función genérica que me diga si un número es múltiplo de otro.

Ejemplo: criba de Eratóstenes

```
class EsMultiploDeY {  
private:  
    int y;  
public:  
    EsMultiploDeY(int y): y(y) { }  
    bool operator()(int x) { return x % y == 0; }  
};
```

Devuelve true si x es múltiplo de y

```
int main() {  
    std::list<int> lista;  
    for (int i = 2; i ≤ 100; i++) { lista.push_back(i); }  
  
    EsMultiploDeY mult_dos(2), mult_tres(3), mult_cinco(5);  
    std::list<int> primos;  
    primos.push_back(lista.front());  
    eliminar(lista, mult_dos);  
    primos.push_back(lista.front());  
    eliminar(lista, mult_tres);  
    primos.push_back(lista.front());  
    eliminar(lista, mult_cinco);  
    ...  
}
```

Objetos función

Esto ya es correcto

Ejemplo: criba de Eratóstenes

```
class EsMultiploDeY {  
private:  
    int y;  
public:  
    EsMultiploDeY(int y): y(y) { }  
    bool operator()(int x) { return x % y == 0; }  
};
```

```
int main() {  
    std::list<int> lista;  
    for (int i = 2; i ≤ 100; i++) { lista.push_back(i); }
```

```
    std::list<int> primos;  
    while (!lista.empty()) {  
        primos.push_back(lista.front());  
        EsMultiploDeY multiples_de_front(lista.front());  
        eliminar(lista, multiples_de_front);  
    }  
}
```

Mientras que la lista inicial siga teniendo elementos pues
el primero lo añado a la lista de primos.
Eliminamos aquellos que sean múltiplos de ese.

¿Para qué sirve un objeto función?

- Cuando queremos pasar una función como parámetro, pero esa función, además de sus argumentos, depende de otros valores.

```
class EsMultiploDeY {  
private:  
    int y;  
public:  
    EsMultiploDeY(int y): y(y) { }  
    bool operator()(int x) { return x % y == 0; }  
};
```

El atributo de la clase.

En este caso querríamos que una función me dijese si un número es múltiplo de otro y ese número es fijo

Sin embargo, tener que crear clases cada vez que queramos crear un objeto función es un poco engorroso. Para ello vemos las **LAMBDA FUNCIONES**.