75.04/95.12 Algoritmos y Programación II

Guía 1, Ejercicio 4: Operador const

Universidad de Buenos Aires - FIUBA

Primer Cuatrimestre de 2018

Explicar las diferencias entre:

- const tipo &r
- tipo const &r
- tipo& const r
- const tipo *r
- tipo const *r
- tipo* const r
- tipo const *const r

Primero, recordaremos de C que const es una palabra reservada para especificar que la variable declarada es constante, es decir, que no se puede modificar por el programa. Afecta al objeto que esté inmediatamente a la izquierda o, en caso de no haber nada, al objeto a la derecha.

Un primer uso es declarar constantes con esto. Tiene la ventaja por sobre las macros creadas con #define en que son entendidas por el compilador y no simplemente reemplazadas, por lo que puede detectar errores y señalarlos mejor.

Luego, es útil en el pasaje de parámetros, ya que además del efecto en sí que tienen de ser tomados como variables de sólo lectura, documentan mejor (se puede ver desde la interfaz que estos parámetros no serán alterados). En particular con referencias sirve para no pasar la variable por copia, así no ocupa espacio doble ni se debe invertir tiempo en generar la copia, pero a su vez asegura no modificar la variable original. Hay que tener en cuenta que si en la función se invoca a su vez otra función con esta variable, ésta deberá también declararla como const. Ya en C normalmente se pasaban por puntero estructuras grandes con este mismo objetivo. Un ejemplo de esto sería:

```
void MiFuncion(const matriz & m)
```

Esta función puede leer la matriz, pero no modificarla, y no la pasa por copia porque si esta fuera de dimensiones grandes, obligaría a tener la original y la copia a la vez en memoria, además de que tardaría en el proceso de crearse llamando al constructor copia.

Similar a esto último está la posibilidad de declarar una función miembro como const (void clase1::metodo()const), lo cual quiere decir que no serán modificados los atributos del objeto que hace la llamada (el apuntado por 'this'). Por ejemplo, un método declarado de esa manera de la clase complejo con atributos real e imaginario no puede asignar this-¿real = a, o real = a; o ninguna forma que afecte estos valores.

Un punto complicado es el uso de const en valores de retorno. Al retornar por referencia, es posible afectar al objeto con una función luego. Devolviendo como referencia constante sólo permite utilizar funciones que lo declaren como const. Supongamos que tenemos los siguientes métodos de la clase1

```
clase1 clase1::Metodo1()
clase1 & clase1::Metodo2()
const clase1 & clase1::Metodo3()
```

El primero devolverá un nuevo objeto de la clase1. Este tendrá que ser asignado afuera, de caso contrario se perderá. El segundo devuelve una referencia, por ejemplo puede devolver el mismo objeto que lo llamó, y esto permitiría encadenarlo de formas como (objeto.Metodo2()).Metodo2() . Con el tercer caso esto no es posible ya que Metodo3 no está declarado como const, por lo que no asegura no modificar a la referencia retornada por el método declarada como constante.

Para las declaraciones del enunciado del ejercicio:

- const tipo &r
- tipo const &r
- const tipo *r
- tipo const *r

Vemos que const tipo y tipo const son idénticas, dado que ambos afectan al dato (const afecta a la palabra a la izquierda más cercana, y si no hay, a la derecha) y quieren decir que, en caso de las referencias, la variable no puede ser modificada, y en el caso de los punteros que no lo puedo modificar desde ese puntero (*p = a; no funciona).

- tipo & const r

Esto sería hacer la referencia en si constante. Esto es una redundancia y no compila.

- tipo * const r

Esto quiere decir que el puntero no puede apuntar a otro dato, si bien este puede ser modificado.

- tipo const * const r

Esto es un puntero constante a un dato constante, ninguno puede ser modificado.