Universidad Nacional del Litoral
Facultad de Ingeniería y Ciencias Hídricas
Departamento de Informática

Ingeniería Informática

PROGRAMACIÓN ORIENTADA A OBJETOS

UNIDAD 4

Sobrecarga de operadores

2011

UNIDAD 4

Sobrecarga de operadores

Ejercicio 4.1

```
class Racional {
    int num, den;
    public:
    Racional(int n, int d){
        num=n;
        den=d;
    };
    int VerNum(){return num;};
    int VerDen(){return den;};
};
```

Dada la clase *Racional* mostrada en el recuadro, implemente sobrecargas para los siguiente operadores:

- El operador + para sumar dos objetos de la clase *Racional*. Recuerde que:

$$\frac{a}{b}+\frac{c}{d}=\frac{a*d+c*b}{b*d}$$

- El operador * para multiplicar dos objetos de la clase *Racional*.
- El operador ++ que permita incrementar en una unidad un número racional.

Finalmente, compruebe el funcionamiento de los operadores con el siguiente programa cliente:

```
int main() {
    Racional a(3, 5), b(2, 3), c(0,1);
    c=a+b;
    cout<<c.VerNum()<<" "<<c.VerDen()<<endl;
    c=a*b;
    c=a+b+c;
    c=a*b*c;
    b=c++;
    a=++c;
    cout<<a.VerNum()<<" "<<a.VerDen()<<endl;
    cout<<b.VerNum()<<" "<<b.VerDen()<<endl;
};
```

Si ocurren errores, intente encontrar su causa.

Ejercicio 4.2

Para la clase *Racional* utilizada en el ejercicio anterior, implemente el operador < para comparar dos números racionales. Haga uso de dicho operador desde un programa cliente.

Ejercicio 4.3

Implemente una clase llamada *Complejo* para representar un número complejo. Sobrecargue los operadores =, + e == para asignar, sumar y comparar respectivamente dos objetos de tipo *Complejo*. Compruebe el funcionamiento de los operadores desde un programa cliente.

Ejercicio 4.4

Tomando como base la *Pila* implementada en el ejercicio 3.2, implemente la sobrecarga de los siguientes operadores.

- El operador += de forma que reciba un entero y y lo apile.
- Otra sobrecarga del operador += para que reciba una segunda pila y apile todos sus elementos en la primera.
- El operador -- (post decremento) para que devuelva el elemento que está en el tope de la pila y lo desapile.
- El operador -= de forma que reciba un entero y quite de la pila esa cantidad de elementos.

Ejercicio 4.5

Para la clase *POOString* desarrollada en la guía 2. Sobrecargue los siguientes operadores:

- El operador [ ] de forma que permita obtener el carácter de la cadena en una posición dada.
- El operador += para concatenar otra cadena a la actual.
- **(OPCIONAL)** El operador + para sumar cadenas, de forma que el siguiente fragmento de código pueda compilar sin errores.

```
POOString a("Programacion");
POOString b("Objetos");
POOString c(a+" Orientada a "+b);
```

Ejercicio 4.6

Para la clase *VectorDinamico* desarrollada en la guía 2 implemente una sobrecarga del operador * que permita multiplicar por un entero a todos los elementos del arreglo. Implemente, además, una sobrecarga del operador ++ que permita duplicar la cantidad de memoria reservada manteniendo los datos que ya estaban en el vector e inicializando al azar los nuevos valores.

Ejercicio 4.7

Realice una búsqueda bibliográfica y averigüe como pueden sobrecargarse los operadores << y >>. Luego implemente:

- Una sobrecarga del operador << para mostrar en pantalla un objeto de la clase *Racional*. El operador debe mostrar el numerador y el denominador separados por el carácter '/'. Responda: la sobrecarga de este operador, ¿debe realizarse dentro o fuera de la clase *Racional*?
- **(OPCIONAL)** Una sobrecarga del operador >> que permita leer desde consola un objeto de tipo *POOString*.
- **(OPCIONAL)** Una sobrecarga del operador << que permita mostrar todos los elementos de un objeto de tipo *VectorDinamico*.

Ejercicio 4.8 (OPCIONAL)

Implemente una clase *Vector2D* con dos números reales como atributos para representar un vector en el plano. Implemente operadores para los productos escalar y vectorial y el producto del vector por un real. Para elegir los operadores a sobrecargar en cada caso, analice la tabla de jerarquía de operadores y seleccione los operadores cuya jerarquía coincida con la que tienen dichos productos en el algebra de vectores convencional.

Ejercicio 4.9 (OPCIONAL)

```
class Ejemplo {
    public:
    operator int() const {
        return 3;
    };
    operator float() const {
        return 3.5;
    };
};

int main() {
    Ejemplo c;
    int i=c;
    float f=c;
    cout<<"int: "<<i<<endl;
    cout<<"float: "<<f<<endl;
    return 0;
}
```

Observe el código del recuadro. ¿Es correcto? Intente ejecutarlo e investigue para qué se utiliza dicha sintaxis.
¿Cómo utilizaría dicha sintaxis para para mostrar objetos de tipo *POOString* de la forma que se muestra en el recuadro de abajo?

```
POOString a("Hola");
cout<<a<<endl;
```

¿Cuál es la diferencia con la sobrecarga del operador <<?

Cuestionario

- ¿Qué es la sobrecarga de operadores?
- ¿Por qué es necesario o que ventajas tiene la sobrecarga de operadores?
- ¿Cuántos argumentos son necesarios para sobrecargar un operador unario? ¿Y uno binario?
- Cuando se sobrecarga un operador para una determinada clase, ¿es necesario definir dicho operador como función miembro de la clase?
- ¿Todos los operadores pueden ser sobrecargados?
- Si sobrecargamos los operadores << o >>, ¿dentro de cuál clase debemos hacerlo?¿Por qué?