UNIVERSIDAD NACIONAL DEL LITORAL
FACULTAD DE INGENIERÍA Y CIENCIAS HÍDRICAS
DEPARTAMENTO DE INFORMÁTICA

Ingeniería en Informática

PROGRAMACIÓN ORIENTADA A OBJETOS

Ingeniería en Inteligencia Artificial

PROGRAMACIÓN II

UNIDAD 4
Sobrecarga de Operadores

Guía de Trabajos Prácticos

EJERCICIOS BÁSICOS

Ejercicio 1

Dada la clase `Racional` mostrada en el recuadro:

```
class Racional {
public:
    Racional(int num, int den) : m_num(num), m_den(den) { }
    int VerNum() const { return m_num; }
    int VerDen() const { return m_den; }
private:
    int m_num, m_den;
};
```

Implemente sobrecargas para los siguiente operadores:

- El operador + para sumar dos objetos de la clase `Racional`.
- El operador $*$ para multiplicar un objeto de tipo `Racional` por un entero.

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

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

```
int main() {
    Racional a(3, 5), b(2, 3);
    Racional aux = a + b;
    cout << aux.VerNum() << '/' << aux.VerDen() << endl;
    aux = a * b;
    cout << p.VerNum() << '/' << p.VerDen() << endl;
}
```

Si ocurren errores de compilación, explique su causa e implemente las correcciones necesarias. Analice: ¿Qué otro operador se utiliza para la clase `Racional` en éste programa cliente? ¿Por qué no es necesario sobrecargarlo?

Ejercicio 2

Para la clase `Racional` utilizada en el ejercicio anterior, implemente los operadores relacionales <, <=, >, >=, == y `!=` para comparar dos números racionales. Haga uso de dichos operadores desde un programa cliente.

Ejercicio 3

Implemente sobrecargas para los operadores >> y << para leer un objeto de tipo `Racional` desde la consola (mediante `cin`) y mostrarlo en pantalla (mediante `cout`). La lectura se debe realizar leyendo el numerador y denominador por separado (es decir, separados por un espacio o salto de línea). El operador << debe mostrar el numerador y el denominador separados por el carácter `'/'`. Analice: la sobrecarga de este operador, ¿debe realizarse dentro o fuera de una clase?

Ejercicio 4

Implemente sobrecargas para pre y post incremento (operador ++) para la clase `Racional` de los ejercicios anteriores. Proponga un programa de prueba donde se note la diferencia entre uno y otro.

Ejercicio 5

Implemente una clase llamada `Complejo` para representar un número complejo. Sobrecargue los operadores <<, >>, +, −, ∗ e == para mostrar, leer, sumar, restar, multiplicar y comparar respectivamente dos objetos de tipo `Complejo`. Compruebe el funcionamiento de los operadores desde un programa cliente.

Ejercicio 6

Implemente una sobrecarga del operador `[ ]` para la clase `Complejo` del ejercicio anterior, que permita obtener las partes real e imaginaria como si fueran 2 elementos en un arreglo:

```
Complejo c(1, 2); // 1+2i
cout << "Parte real: " << c[0] << endl; // muestra 1
cout << "Parte imag: " << c[1] << endl; // muestra 2
```

Explique: ¿Podría su sobrecarga utilizarse para modificar las partes del número complejo?

```
Complejo c;
cout << "Ingrese la parte real: ";
cin >> c[0];
cout << "Ingrese la parte imaginaria: ";
cin >> c[1];
```

CUESTIONARIO

1. ¿Qué es la sobrecarga de operadores?
2. ¿Por qué es necesario o que ventajas tiene la sobrecarga de operadores?
3. ¿Cuántos argumentos son necesarios para sobrecargar un operador unario? ¿Y uno binario?
4. ¿Por qué cree usted que conviene o se acostumbra a que el paso de parámetros de objetos a funciones se haga por referencia?
5. ¿Para qué sirve la palabra clave `const` aplicada a un parámetro pasado por referencia? ¿Qué función cumple dicha palabra reservada cuando es aplicada a una función miembro?
6. Cuando se sobrecarga un operador para una determinada clase, ¿es necesario definir dicho operador como función miembro de la clase?
7. ¿Todos los operadores pueden ser sobrecargados?
8. Si intentáramos sobrecargar los operadores << o >>, ¿dentro de cuál clase deberíamos hacerlo? ¿Por qué?

EJERCICIOS ADICIONALES

Ejercicio 1

Implemente una clase llamada `Vector2D` con dos números reales como atributos para representar un vector en el plano (componentes *x* e *y*). 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 álgebra de vectores convencional.

Ejercicio 2

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 asignar un objeto de tipo `VectorDinamico` a otro objeto del mismo tipo, copiando los elementos. Pruebe los operadores sobrecargados en un programa cliente, luego, comente la sobrecarga del operador `=` para que no sea compilada y explique lo que sucede. Implemente una sobrecarga del operador `[ ]` para ver y modificar los elementos del vector, que además verifique si el índice que recibe es correcto, y muestre un mensaje de error en caso contrario.

Ejercicio 3

Observe el código del recuadro. ¿Es correcto? Intente ejecutarlo e investigue para qué se utiliza dicha sintaxis.

```
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;
}
```