Autor: Martin Alejandro García

Sobrecarga de operadores

Los operadores como +, -, == y << son herramientas fundamentales que permiten realizar diversas operaciones en los datos. Sin embargo, cuando trabajamos con objetos en C++, los operadores estándar no siempre se comportan como necesitamos. Para solucionar esto, C++ permite modificar el comportamiento de ciertos operadores mediante un mecanismo llamado **sobrecarga de operadores**.

La sobrecarga de operadores permite redefinir cómo funcionan los operadores en el contexto de nuestras propias clases y objetos. Esto es particularmente útil cuando queremos que los objetos de nuestras clases se comporten de manera intuitiva al usarlos en expresiones o comparaciones, como si fueran tipos de datos primitivos.

Por ejemplo, si tenemos una clase Persona, y queremos comparar dos objetos de esta clase para ver si representan a la misma persona (por un atributo como el nombre o la edad), no podemos usar directamente el operador == como lo haríamos con números. Aquí es donde entra en juego la **sobrecarga del operador ==**, que nos permite definir cómo deben compararse los objetos de esa clase.

Este concepto es fundamental en la programación orientada a objetos, ya que ayuda a que el código sea más limpio y fácil de entender, al permitir que los objetos se comporten de manera similar a los tipos de datos primitivos.

Para ejemplificar vamos a tomar la clase Persona, de manera simplificada y ver distintos comportamientos.

```
class Persona
{
    public:
        Persona(const char* nombre, const char* apellido, int edad);
        void setNombre(const char* nombre);
        void setApellido(const char* apellido);
        void setEdad(int edad);
        void setSueldo(float sueldo);
        void mostrarDatos();

        const char* getNombre();
        const char* getApellido();
        int getEdad();
        float getSueldo();

    private:
        char _nombre[50];
        char _apellido[50];
        int _edad;
        float _sueldo;
};
```

Ahora bien, con respecto a su uso, podemos usarlas como miembro de la clase o por fuera.

Función miembro: Cuando el operador actúa directamente sobre el objeto que invoca la función. En este caso, el primer parámetro (el objeto a la izquierda del operador) es implícito, ya que es el objeto que llama al operador.

```
bool Persona::operator==(const Persona p2)
```

Función no miembro: Útil cuando el operador debe actuar de manera simétrica (por ejemplo, cuando el operador funciona con objetos de distintas clases).

```
std::ostream& operator<<(std::ostream& os, Persona p);
```

Operadores Sobrecargables

En C++, casi todos los operadores pueden sobrecargarse. Algunos de los más comunes incluyen:

- Aritméticos: +, -, *, /
- Comparación: ==, !=, <, >, <=, >=
- Asignación: =, +=, -=, *=, /=
- Acceso: [], (), ->
- Entrada/Salida: <<, >>

(En este apunte nos vamos a centrar en los operadores Aritméticos y de Comparación)

Sin embargo, hay algunos operadores que **no se pueden sobrecargar**, como:

- El operador de resolución de ámbito :: (doble dos puntos)
- El operador de miembro . (punto)
- El operador condicional ternario ?:

Reglas Importantes

- **Preservar el comportamiento esperado**: La sobrecarga de operadores debe respetar el significado semántico de los operadores. Por ejemplo, sobrecargar el operador + debe ser coherente con la operación de suma.
- **No cambiar la aridad**: No se puede cambiar el número de operandos de un operador. Por ejemplo, el operador + siempre debe tener dos operandos.
- **No crear operadores nuevos**: Solo puedes sobrecargar los operadores existentes en C++.

Retorno de Tipos Adecuados

El tipo de retorno de la función debe ser coherente con el operador que estás sobrecargando. Por ejemplo, si sobrecargas ==, el tipo de retorno será bool, ya que el operador de igualdad produce un resultado booleano.

Ahora si, teniendo en cuenta estas cuestiones nos enfocamos en sobrecargar operadores en la clase Persona.

```
class Persona
{
    public:
        Persona(const char* nombre, const char* apellido, int edad);
        void setNombre(const char* nombre);
        void setApellido(const char* apellido);
        void setEdad(int edad);
        void setSueldo(float sueldo);
        void mostrarDatos();

        const char* getNombre();
        const char* getApellido();
        int getEdad();
        float getSueldo();

        bool operator==(const Persona p2);
        bool operator<(const Persona p2);
        int operator+(const Persona p2);
        void operator=(const Persona p2);
    private:
        char _nombre[50];
        char _apellido[50];
        int _edad;
        float _sueldo;
};
```

Este fragmento de código nos muestra el archivo cabecera o .h de la clase Persona, donde están declarados sus propiedades/atributos y métodos.

Operadores de comparación

```
bool Persona::operator==(const Persona p2)
{
    return this->_edad == p2._edad;
}
```

En este caso, el operador == es sobrecargado para que compare la edad de la persona que lo invoca con la que se envía por parámetro. Veamos otro ejemplo pero con otro operador:

```
bool Persona::operator<(const Persona p2)
{
    return strcmp(this->_nombre, p2._nombre) < 0;
}
```

Este ejemplo nos muestra como sobrecargando el operador < (menor) podemos comparar dos personas según su nombre mediante la función string compare (strcmp). Si el nombre del objeto persona que lo invoca fuera Brian y el que se manda por parámetro para comparar fuera Daniel retornaría true ya que la B está antes en el alfabeto.

```
int main() {
    Persona lalo("Lalo", "Landa", 10);
    Persona nick("Nick", "Riviera", 45);
    Persona bart("Bart", "Simpson", 10);
    Persona rafa("Rafa", "Gorgory", 8);

    std::cout << "Sobre carga de operadores de comparacion (== y <)\n";
    if (lalo == nick) {
        std::cout << "Tienen la misma edad";
    }
    else {
        std::cout << "Tienen distinta edad";
    }

    std::cout << "\n";

    if (lalo < bart) {
        std::cout << "Lalo va primero en la lista";
    }
    else {
        std::cout << "Bart va primero en la lista";
    }
```

Operadores de Asignación

```
int Persona::operator+(const Persona p2)
{
    return this->_edad + p2._edad;
}
```

Si quisiéramos sumar las edades de dos personas se podría sobrecargar el operador + (suma) y de esta manera, al sumar dos objetos de la misma clase tendríamos el resultado de sumar las dos edades.

```
void Persona::operator=(const Persona p2)
{
    this->_edad = p2._edad;
}
```

Siguiendo con el ejemplo de la edad también podríamos igualar las edades de dos personas sobrecargando el operador de asignación = (igual)

```
std::cout << "Sobre carga operadores aritmeticos (+)\n";
int suma = bart + lalo;
std::cout << "Si sumo las edades de Bart y Lalo obtengo: " << suma;

std::cout << "Sobre carga de operadores de asignacion (=)\n";
lalo = nick;
std::cout << "Ahora la edad de Lalo es " << lalo.getEdad() << " al igual que " << nick.getNombre();
```

El siguiente enlace los lleva al repositorio de GitHub en el que se encuentra el código completo de los ejemplos.

Repositorio de GitHub