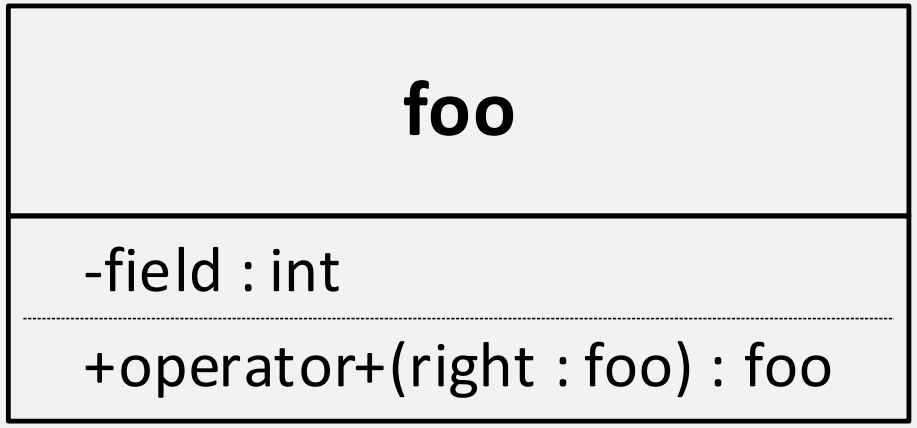
OPERATORS AS MEMBER FUNCTIONS

Definition and Call

Delroy A. Brinkerhoff

CLASS WITH AN OVERLOADED OPERATOR

UML



C++

```
class foo
{
    private:
        int field;
    public:
        foo operator+(foo right);
};
```

FUNCTION DEFINITION

IN CLASS

```
class foo
{
    private:
        int field;
    public:
        foo operator+(foo right)
        {
            return foo(field + right.field);
        }
};
```

EXTERNAL

```
class foo
{
    private:
        int field;
    public:
        foo operator+(foo right);
};


foo foo::operator+(foo right)
{
    return foo(field + right.field);
}
```

FUNCTION CALL NOTATIONS

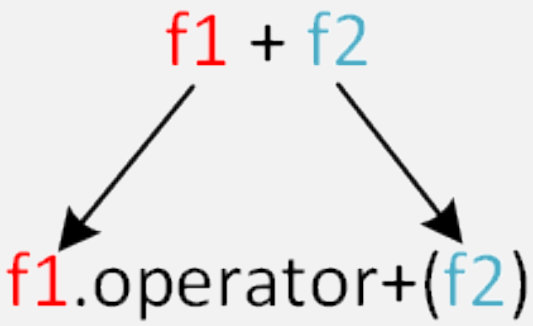
```
foo foo::operator+(foo right)
{
    return foo(field + right.field);
}
```

- `foo f1;`
- `foo f2;`
- `foo f3;`

- `f3 = f1.operator+(f2);`
- `f3 = f1 + f2;`

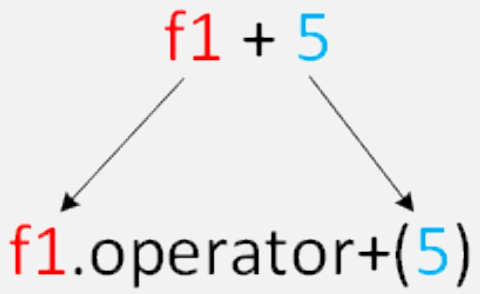
THE RELATIONSHIP BETWEEN OPERANDS AND ARGUMENTS

- The left-hand operand is *always* the object bound to the function call (i.e., the `this` or the implicit argument)
- The right-hand operand is *always* the argument in the parentheses (i.e., the explicit argument)



OVERLOADING OVERLOADED OPERATORS

PROBLEM: ADDING OBJECTS AND FUNDAMENTAL TYPES



SOLUTION: OVERLOADING OVERLOADED OPERATORS

```
foo foo::operator+(int right)
{
    return foo(field + right);
}
```

THE fraction CLASS

```
class fraction
{
    private:
        int     numerator;
        int     denominator;

    public:
        fraction(int n = 0, int d = 1);
        fraction operator+(fraction f2);
        fraction operator+(int i);
};
```

IN-CLASS FUNCTION DEFINITION

```
class fraction
{
    public:

        fraction operator+(fraction f2)
        {
            int    n = numerator * f2.denominator + f2.numerator * denominator;
            int    d = denominator * f2.denominator;

            return fraction(n, d);
        }
};
```

EXTERNAL FUNCTION DEFINITION

```
fraction fraction::operator+(int i)
{
    int    n = numerator + denominator * i;

    return fraction(n, denominator);
}
```