21.6 — Overloading unary operators +, -, and !

learncpp.com/cpp-tutorial/overloading-unary-operators/

Overloading unary operators

Unlike the operators you've seen so far, the positive (+), negative (-) and logical not (!) operators all are unary operators, which means they only operate on one operand. Because they only operate on the object they are applied to, typically unary operator overloads are implemented as member functions. All three operators are implemented in an identical manner.

Let's take a look at how we'd implement operator- on the Cents class we used in a previous example:

```
#include <iostream>

class Cents
{
private:
    int m_cents {};

public:
    Cents(int cents): m_cents{cents} {}

    // Overload -Cents as a member function
    Cents operator-() const;

    int getCents() const { return m_cents; }
};

// note: this function is a member function!
Cents Cents::operator-() const
{
    return -m_cents; // since return type is a Cents, this does an implicit
conversion from int to Cents using the Cents(int) constructor
}

int main()
{
    const Cents nickle{ 5 };
    std::cout << "A nickle of debt is worth " << (-nickle).getCents() << " cents\n";

    return 0;
}
```

This should be straightforward. Our overloaded negative operator (-) is a unary operator implemented as a member function, so it takes no parameters (it operates on the *this object). It returns a Cents object that is the negation of the original Cents value. Because operator- does not modify the Cents object, we can (and should) make it a const function (so it can be called on const Cents objects).

Note that there's no confusion between the negative operator- and the minus operator- since they have a different number of parameters.

Here's another example. The ! operator is the logical negation operator -- if an expression evaluates to "true", operator! will return false, and vice-versa. We commonly see this applied to boolean variables to test whether they are true or not:

```
if (!isHappy)
    std::cout << "I am not happy!\n";
else
    std::cout << "I am so happy!\n";
```

For integers, 0 evaluates to false, and anything else to true, so operator! as applied to integers will return true for an integer value of 0 and false otherwise.

Extending the concept, we can say that operator! should evaluate to true if the state of the object is "false", "zero", or whatever the default initialization state is.

The following example shows an overload of both operator- and operator! for a user-defined Point class:

```cpp
#include <iostream>

class Point
{
private:
    double m_x {};
    double m_y {};
    double m_z {};

public:
    Point(double x=0.0, double y=0.0, double z=0.0):
        m_x{x}, m_y{y}, m_z{z}
    {
    }

    // Convert a Point into its negative equivalent
    Point operator- () const;

    // Return true if the point is set at the origin
    bool operator! () const;

    double getX() const { return m_x; }
    double getY() const { return m_y; }
    double getZ() const { return m_z; }
};

// Convert a Point into its negative equivalent
Point Point::operator- () const
{
    return { -m_x, -m_y, -m_z };
}

// Return true if the point is set at the origin, false otherwise
bool Point::operator! () const
{
    return (m_x == 0.0 && m_y == 0.0 && m_z == 0.0);
}

int main()
{
    Point point{}; // use default constructor to set to (0.0, 0.0, 0.0)

    if (!point)
        std::cout << "point is set at the origin.\n";
    else
        std::cout << "point is not set at the origin.\n";

    return 0;
}
```

The overloaded operator! for this class returns the Boolean value “true” if the Point is set to the default value at coordinate (0.0, 0.0, 0.0). Thus, the above code produces the result:

```
point is set at the origin.
```

Quiz time

1. Implement overloaded unary `operator+` for the Point class. Unary `operator+` just returns its operand (it does not make negative values positive).

Show Solution

Next lesson
21.7Overloading the comparison operators
Back to table of contents
Previous lesson
21.5Overloading operators using member functions