Document number: NXXXX
Date: 2014-08-28
Project: Programming Language C++,
Language Evolution Working Group
Reply to: Marcel Wid
<marcel.wid@ods-solutions.de>

Explicitly defaulted binary operators

Contents

I. Introduction

Since C++11 it is possible to explicitly default some special member functions of a class. We propose to extend the possibility to explicitly default the binary operators `operator*`, `operator/`, `operator%`, `operator+`, `operator-`, `operator>>`, `operator<<`, `operator&`, `operator^` and `operator|`. All these operators can be defined using the corresponding compound assignment operator. We also propose to explicitly default the postfix incerement `operator++(int)` and postfix decrement `operator--(int)`. They can be defined using the corresponding prefix form.

II. Motivation and Scope

2.1 Binary Operators

It is good and common practice defining the assignment version `a+=b` when defining `a+b` for a class type and to let both have the same semantics. The canonical way to achieve this is to define `operator+` in terms of `operator+=`.

```
class T
{
    // ...
};

T& T::operator+=(const T& other)
{
    // implementation
```

```
    return *this;
}

T operator+(T lhs, const T& rhs)
{
    return lhs += rhs;
}
```

Listing 1: Definition of `operator+` using `operator+=`

Note that `operator+` takes it first argument by value in order to make use of move semantics. This results in a lot of boilerplate code, is needlessly verbose and error prone. Hence we propose to allow the following syntax, which is equivalent to the above.

```
class T
{
    // ...
};

T& T::operator+=(const T& other)
{
    // implementation
    return *this;
}

T operator+(T lhs, const T& rhs) = default;
```

Listing 2: Defining `operator+` as explicitly defaulted

2.2 Increment and Decrement

It is also good practice to define the postincrement operator in terms of the preincrement one and the same for decrement operators.

```
class T
{
    // ...
};

T& T::operator++()
{
    // implementation
    return *this;
}

T T::operator++(int)
{
    T old{*this};
    ++(*this);
    return old;
}
```

Listing 3: Definition of postincrement using preincrement

To release the programmer from writing such mechanical lines of code, we propose to explicitly default `operator++(int)`. The following code fragment is equivalent to the above one.

```
class T
{
    // ...
};

T& T::operator++()
{
```

```
    // implementation
    return *this;
}

T T::operator++(int) = default;
```

Listing 4: Defining `operator++(int)` as explicitly defaulted

III. Design Decisions

We refer the reader to Sutter & Alexandrescu, item 27 and 28, for a general discussion of implementing these operators. It should be notet that this proposal does not break any existing code, since this new feature has to be explicitly "opt in". Moreover, no new keyword has to be added to the core language. Only the set of functions which can be explicitly defaulted is enlarged.

IV. Technical Specifications

4.1 Informal Specification

4.1.1 Binary Operators

Let `@` be one of the binary operators `*`, `/`, `\%`, `+`, `-`, `>>`, `<<`, `&`, `^` or `|`. The following code

```
class T
{
    // ...
};

T operator@(T lhs, const T& rhs) = default;
```

is rewritten to

```
class T
{
    // ...
};

T operator@(T lhs, const T& rhs)
{
    return lhs @= rhs;
}
```

4.1.2 Increment and Decrement

Let `@` be one of `+` or `-`. The following code

```
class T
{
    // ...
};

T T::operator@@(int) = default;
```

is rewritten to

```
class T
{
    // ...
};

T T::operator@@(int)
{
    T old{*this};
    @@(*this);
    return old;
}
```

4.2 Proposed Wording

to be added ...

V. Acknowledgments

References

C++ Standards Committee and others (2011). ISO/IEC 14882: 2011, Standard for Programming Language C++. Technical report, ISO/IEC. http://www.open-std.org/jtc1/sc22/wg21.

Sutter, H. & Alexandrescu, A. (2004). *C++ coding standards: 101 rules, guidelines, and best practices*. Pearson Education.