27. Отдавайте предпочтение каноническим формам арифметических операторов и операторов присваивания

Резюме

Если можно записать a+b, то необходимо, чтобы можно было записать и a+=b. При определении бинарных арифметических операторов одновременно предоставляйте и их присваивающие версии, причем делайте это с минимальным дублированием и максимальной эффективностью.

Обсуждение

В общем случае для некоторого бинарного оператора @ (+, -, * и т.д.) вы должны также определить его присваивающую версию, так чтобы a@=b и a=a@b имели один и тот же смысл (причем первая версия может быть более эффективна). Канонический способ достижения данной цели состоит в определении @ посредством @= следующим образом:

```
T& T::operator@=( const T& ) {
    // ... реализация ...
    return *this;
}

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

Эти две функции работают в тандеме. Версия оператора с присваиванием выполняет всю необходимую работу и возвращает свой левый параметр. Версия без присваивания создает временную переменную из левого аргумента, и модифицирует ее с использованием формы оператора с присваиванием, после чего возвращает эту временную переменную.

Обратите внимание, что здесь operator@ — функция-не член, так что она обладает желательным свойством возможности неявного преобразования как левого, так и правого параметра (см. рекомендацию 44). Например, если вы определите класс String, который имеет неявный конструктор, получающий аргумент типа char, то оператор operator+(const String&, const String&), который не является членом класса, позволяет осуществлять операции как типа char+String, так и String+char; функция-член String::operator+(const String&) позволяет использовать только операцию String+char. Реализация, основной целью которой является эффективность, может определить ряд перегрузок оператора operator@, не являющихся членами класса, чтобы избежать увеличения количества временных переменных в процессе преобразований типов (см. рекомендацию 29).

Также делайте не членом функцию operator@= везде, где это возможно (см. рекомендацию 44). В любом случае, все операторы, не являющиеся членами, должны быть помещены в то же пространство имен, что и класс T, так что они будут легко доступны для вызывающих функций при отсутствии каких-либо сюрпризов со стороны поиска имен (см. рекомендацию 57).

Как вариант можно предусмотреть оператор operator@, принимающий первый параметр по значению. Таким образом вы обеспечите неявное копирование компилятором, что обеспечит ему большую свободу действий по оптимизации:

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

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

Еще один вариант — оператор `operator@`, который возвращает `const`-значение. Эта методика имеет то преимущество, что при этом запрещается такой не имеющий смысла код, как `a+b=c`, но в этом случае мы теряем возможность применения потенциально полезных конструкций наподобие `a = (b+c).replace(pos,n,d)`. А это весьма выразительный код, который в одной строчке выполняет конкатенацию строк `b` и `c`, заменяет некоторые символы и присваивает полученный результат переменной `a`.

Примеры

Пример. Реализация += для строк. При конкатенации строк полезно заранее знать длину, чтобы выделять память только один раз:

```
String& String::operator+=( const String& rhs ) {
    // ... Реализация ...
    return *this;
}

String operator+( const String& lhs, const String& rhs ) {
    String temp;                    // Изначально пуста
    // Выделение достаточного количества памяти
    temp.Reserve(lhs.size()+rhs.size());
    // Конкатенация строк и возврат
    return (temp += lhs) += rhs;
}
```

Исключения

В некоторых случаях (например, оператор `operator*=` для комплексных чисел), оператор может изменять левый аргумент настолько существенно, что более выгодным может оказаться реализация оператора `operator*=` посредством оператора `operator*`, а не наоборот.

Ссылки

[Alexandrescu03a] • [Cline99] §23.06 • [Meyers96] §22 • [Sutter00] §20