15. Активно используйте const

Резюме

const — ваш друг: неизменяемые значения проще понимать, отслеживать и мотивировать, т.е. там, где это целесообразно, лучше использовать константы вместо переменных. Сделайте const описанием по умолчанию при определении значения — это безопасно, проверяемо во время компиляции (см. рекомендацию 14) и интегрируемо с системой типов C++. Не выполняйте преобразований типов с отбрасыванием const кроме как при вызове некорректной с точки зрения употребления const функции (см. рекомендацию 94).

Обсуждение

Константы упрощают код, поскольку вам достаточно только один раз взглянуть на ее определение, чтобы знать, чему она равна везде. Рассмотрим такой код:

```
void Fun( vector<int>& v ) {
    // ...
    const size_t len = v.size();
    // ... и еще 30 строк ...
}
```

Увидев такое определение len, вы получаете надежную информацию о семантике этой константы в пределах области ее видимости (в предположении, что код не устраняет ее константность, чего он делать не должен, как вы узнаете далее): это информация о длине v в определенной точке программы. Взглянув на одну строку, вы получили всю необходимую информацию для всей области видимости. Если бы переменная len не была определена как const, она могла бы быть позже изменена — непосредственно или косвенно.

Заметим, что описание const не является глубоким. Чтобы понять что имеется в виду, рассмотрим класс C, который имеет член типа X*. В объекте C, который является константой, член X* также является константой, но сам объект X, на который он указывает, константой не является (см. [Saks99]).

Логическую константность следует реализовывать с использованием членов, описанных как mutable. Когда константная функция-член класса оправданно требует модификации переменной-члена (т.е. когда эта переменная не влияет на наблюдаемое состояние объекта, например, если это кэшированные данные), объявите эту переменную-член как mutable. Заметим, что если все закрытые члены скрыты с использованием идиомы Pimpl (см. рекомендацию 43), описание mutable не является необходимым ни для кэшированной информации, ни для неизменного указателя на нее.

Модификатор const напоминает вирусное заболевание — появившись в вашем коде один раз, он приведет к необходимости соответствующего изменения сигнатур функций, которые еще не являются корректными в плане использования const. Это как раз не ошибка, а хорошее свойство, существенно увеличивающее мощь модификатора const, который еще не так давно был достаточно заброшен, а его возможности не вполне поняты и оценены. Переделка существующего кода для его корректности в плане использования const требует усилий, но они стоят того и даже позволяют выявить скрытые ошибки.

Корректное применение const дает отличные результаты и повышает эффективность. Чрезвычайно важно правильно и последовательно использовать модификатор const в ваших программах. Понимание того, как и где изменяется состояние программы, особенно необходимо, а модификатор const по сути документирует непосредственно в коде программы, где именно компилятор может помочь вам в этом. Правильное употребление const поможет вам лучше разобраться с вопросами проектирования и сделать ваш код более надежным и безо-

пасным. Если вы выяснили, что некоторую функцию-член невозможно сделать константной, значит, вы более детально разобрались с тем, как, где и почему эта функция модифицирует состояние объекта. Кроме того, вы сможете понять, какие члены-данные объединяют физическую и логическую константность (см. приведенные ниже примеры).

Никогда не прибегайте к преобразованию константного типа в неконстантный, кроме случаев вызова функции, некорректной в плане использования модификатора `const` (не модифицирующей параметр, который тем не менее описан как неконстантный), а также такого редкого случая, как способ замены `mutable` в старом компиляторе, не поддерживающем эту возможность.

Примеры

Пример. Избегайте `const` в объявлениях функций, принимающих параметры по значению. Два следующих объявления абсолютно эквивалентны:

```
void Fun( int x );

void Fun( const int x ); // Объявление той же самой функции:
                         // const здесь игнорируется
```

Во втором объявлении модификатор `const` избыточен. Мы рекомендуем объявлять функции без таких высокоуровневых модификаторов `const`, чтобы тот, кто читает ваши заголовочные файлы, не был дезориентирован. Однако использование такого модификатора имеет значение в *определении* функции и его применение может быть оправдано с точки зрения обнаружения непреднамеренного изменения переданного параметра:

```
void Fun( const int x ) { // Определение функции Fun
    // ...
    ++x; // Ошибка: нельзя изменять константное значение
    // ...
}
```

Ссылки

[Allison98] §10 • [Cline99] §14.02-12 • [Dewhurst03] §6, §31-32, §82 • [Keffer95] pp. 5-6 • [Koenig97] §4 • [Lakos96] §9.1.6, §9.1.12 • [Meyers97] §21 • [Murray93] §2.7 • [Stroustrup00] §7.2, §10.2.6, §16.3.1 • [Sutter00] §43