Многоликий const

C++*

Ключевое слово const — одно из самых многозначных в C++. Правильно использование const позволяет организовать множество проверок ещё на этапе компиляции и избежать многих ошибок из числа тех, которые бывает трудно найти при помощи отладчиков и/или анализа кода.

Первая половина заметки рассчитана скорее на начинающих (надеюсь мнемоническое правило поможет вам запомнить, где и для чего используется const), но, возможно, и опытные программисты смогут почерпнуть интересную информацию о перегрузке методов по const.

Константы и данные

Самый простой случай — константные данные. Возможно несколько вариантов записи:

```
const int i(1);
int const j(1);
int const k=1;
```

Все они правильные и делают одно и тоже — создают переменную, значение которой изменить нельзя.

```
const int k=1;
k = 7; // <-- ошибка на этапе компиляции!
```

Константы и указатели

При использовании const с указателями, действие модификатора может распространяться либо на значение указателя, либо на данные на которые указывает указатель.

Работает (const относится к данным):

```
const char * a = "a";
a="b";
```

Тоже самое и тоже работает:

```
char const * a = "a";
a="b";
```

А вот это уже не работает:

```
char * const a = "a";
a="b"; // <-- не работает
```

Если бы операция присвоения изменяла бы не указатель, а данные:

```
*a = 'Y';
```

то ситуация была бы диаметрально противоположной.

Существует мнемоническое правило, позволяющее легко запомнить, к чему относится const. Надо провести черту через "*", если const слева, то оно относится к значению данных; если справа — к значению указателя.

Ну и конечно, const можно написать дважды:

```
const char * const s = "data";
```

Константы и аргументы/результаты функций

С функциями слово const используется по тем же правилам, что при описании обычных данных.

Константы и методы (перегрузка)

А вот с методами есть одна тонкость.

Во-первых, для методов допустимо использование const, применительно к this. Синтаксис таков:

```
class A {
private:
  int x;
public:
  void f(int a) const {
    x = a; // <-- не работает
  }
};
```

Кроме того, этот const позволяет перегружать методы. Таким образом, вы можете писать оптимизированные варианты методов для константных объектов.

Поясняю:

```
class A {
private:
  int x;
public:
  A(int a) {
    x = a;
    cout << "A(int) // x=" << x << endl;
  }
  void f() {
    cout << "f() // x=" << x << endl;
  }
  void f() const {
    cout << "f() const // x=" << x << endl;
  }
};
int main() {
  A a1(1);
  a1.f();
  A const a2(2);
  a2.f();
  return 0;
}
```

Результат:

```
A(int) // x=1
f() // x=1
A(int) // x=2
f() const // x=2
```

То есть для константного объекта (с x=2) был вызван соответствующий метод.

Осталось только добавить, что если вы планируете использовать const-объекты, то вам надо обязательно реализовать const-методы. Если вы в этом случае не реализуете не-const-методы, то во всех случаях будут молча использоваться const-методы. Одним словом, const лучше использовать там, где это возможно.