<https://habrahabr.ru/post/243981/>

C++14 для Qt программистов

<http://woboq.com/blog/cpp14-in-qt.html>

* [Qt](https://habrahabr.ru/hub/qt_software/),
* [C++](https://habrahabr.ru/hub/cpp/)
* Перевод

В этой статье описывается каким образом изменения, принесенные стандартом С++14, отразились или могут отразиться на разработке Qt приложений. Данная статья ориентирована не только на Qt программистов, но также на всех тех, кому интересно развитие С++. Автор оригинала — Olivier Goffart, являющийся одним из разработчиков Qt moc (meta-object compiler).  
  
  
sdfsf

Обобщенные лямбда-функции

В С++11 были введены лямбда-функции, и Qt5 позволяет использовать их в сигналах. C++14 упрощает использование лямбда-функций, так как теперь тип аргументов может быть выведен автоматически, то есть стало возможным использование auto в качестве типа параметра вместо того, чтобы явно описывать этот тип:

connect(sender, &Sender::valueChanged, [=](const auto &newValue) {

receiver->updateValue("senderValue", newValue);

});

Лямбда-функция представляет собой функтор с реализованным оператором operator(). В обобщенных лямбда-функциях этот оператор объявлен как шаблонная функция. Я сделал изменения, которые поддерживают такие функторы и эти изменения были включены в Qt 5.1. C++14 также добавляет возможность захвата не только переменных, но и выражений:

connect(sender, &Sender::valueChanged, [receiver=getReceiver()](const auto &newValue) {

receiver->updateValue("senderValue", newValue);

});

Смягчение требований к константным выражениям

В С++11 было введено новое ключевое слово constexpr. В Qt 4.8 был введен новый макрос Q\_DECL\_CONSTEXPR, который разворачивается в constexpr, если это слово поддерживается компилятором. В Qt 5 этот макрос используется для большого количества функций, где это только представляется возможным.  
В С++14 были смягчены требования, предъявляемые к константным выражениям. С++11 позволял использовать constexpr только с единственным оператором возврата и только в функциях-членах с модификатором const. С++14 позволяет намного больше, лишь бы вычисление могло происходить во время компиляции. 

*/\**

*Эта функция не скомпилируется в С++11, потому что состоит из нескольких компонентов, содержит цикл и внутреннюю переменную.*

*Но в С++14 это разрешено*

*\*/*

constexpr int myFunction(int v) {

int x = 1;

while (x < v\*v)

x\*=2;

return x;

}

Функции-члены класса, объявленные как constexpr в С++11 автоматически трактуются как константные, то есть, не изменяющие поля класса. В С++14 неконстантная функция-член класса тоже может быть constexpr. Результатом такого изменения стало то, что функции-члены классов, объявленные constexpr, но не имеющие явно указанного модификатора const, стали неконстантными в С++14, а это означает несовместимость на уровне бинарных файлов. К счастью, в Qt макрос Q\_DECL\_CONSTEXPR явно объявлял все функции-члены классов константными, поэтому никакого нарушения бинарной совместимости при его использовании нет.  
Итак, теперь мы можем вычислять во время компиляции неконстантные функции классов, такие, например, как operator=. Для этого в Qt 5.5 будет введен новый макрос Q\_DECL\_RELAXED\_CONSTEXPR, который будет разворачиваться в constexpr, если компилятор в режиме С++14. 

Небольшие изменения в С++14

Стандарт С++14 привнес некоторое количество небольших изменений, цель которых — сделать разработку более удобной. Эти изменения не имеют непосредственного влияния на Qt, но вполне могут быть использованы в Ваших программах, если используется компилятор с поддержкой С++14.

Разделители разрядов чисел

Если нужно определить большую константу в коде, можно использовать апостроф в качестве разделителя разрядов:

int i = 123'456'789;

Двоичные константы

В С++ можно определять десятичные, восьмеричные (начинающиеся с 0), и шестнадцатеричные (начинающиеся с 0x) константы. Теперь появилась возможность определять и двоичные константы, используя префикс 0b: 

int i = 0b0001'0000'0001;

Автоматический вывод типа возвращаемого значения

Если у Вас есть встроенная (inline) функция, то Вы можете использовать ключевое слово auto в качестве указания возвращаемого типа, его теперь можно не указывать явно. Компилятор сам его выведет:

*// возвращаемый тип будет выведен как 'int'*

auto sum(int a, int b) { return a+b; }

Это, к сожалению, не поддерживается для Qt слотов или так называемых invocable методов, так как Qt moc не в состоянии сам определить возвращаемый тип.

Шаблонные переменные

Раньше было возможным сделать шаблонную функцию или класс. Сейчас можно сделать шаблонной и просто переменную.

template<typename T> const T pi = 3.141592653589793;

*/\*...\*/*

float f = pi<float>;

double d = pi<double>;

Инициализация структур

В С++11 стало возможно инициализировать структуру, у которой нет определенного пользователем конструктора, списком инициализации (значения полей в фигурных скобках), а также появилась возможность присваивать нестатическим полям класса значения по умолчанию прямо в определении класса. Но в С++11 нельзя было использовать сразу оба этих варианта инициализации. В С++14 теперь можно. Этот код будет работать именно так, как и ожидается:

struct MyStruct {

int x;

QString str;

bool flag = false;

QByteArray str2 = "something";

};

*// ...*

*// не скомпилируется в C++11 потому что MyStruct не POD*

MyStruct s = { 12, "1234", true };

Q\_ASSERT(s.str2 == "something");

Квалификаторы ссылок для методов классов

Это на самом деле было привнесено не в С++14, а еще в С++11, но мы начали использовать эти квалификаторы только в Qt5, и я не упоминал о них в предыдущих постах, поэтому поговорим о них сейчас.  
Рассмотрим следующий код:

QString lower = QString::fromUtf8(data).toLower();

Здесь fromUtf8 возвращает временную переменную. Было бы неплохо, если бы метод toLower использовал уже выделенную память для этой временной переменной и выполнил в ней необходимые преобразования. Именно для подобных случаев и были введены квалификаторы ссылок для функций-членов классов.  
Упрощенный код из qstring.h:

class QString {

public:

*/\* ... \*/*

QString toLower() const &

{ */\* ... возвращает копию со всеми символами в нижнем регистре ... \*/* }

QString toLower() &&

{ */\* ... выполняет преобразования в исходной строке ... \*/* }

*/\* ... \*/*

};

Обратите внимание на '&' и '&&' в конце методов toLower. Это квалификаторы ссылок и позволяют перегрузить функцию в зависимости от типа, на который указывает 'this', таким же образом, как квалификатор const позволяет перегрузить метод в зависимости от константности 'this'. В случае, когда toLower вызывается для временной переменной (rvalue) будет выбран второй метод (который с &&) и проведет изменения строки, не копируя ее.  
Функции, которые были улучшены с помощью этих квалификаторов в Qt 5.4: QString::toUpper, QString::toLower, QString::toCaseFolded, QString::toLatin1, QString::toLocal8Bit, QString::toUtf8, QByteArray::toUpper, QByteArray::toLower, QImage::convertToFormat, QImage::mirorred, QImage::rgbSwapped, QVersionNumber::normalized, QVersionNumber::segment

Изменения в стандартной библиотеке

С++11 и С++14 добавили много конструкций в стандартную библиотеку, которые во многом перекликаются с имеющимися конструкциями в QtCore. В Qt стандартная библиотека используется очень мало. Мы вообще не хотим, чтобы стандартная библиотека была частью ABI. Это позволит оставаться бинарно совместимыми даже если стандартная библиотека изменится (например libstdc++ и libcpp). Также Qt до сих пор поддерживает некоторые старые платформы, на которых нет стандартной библиотеки С++11. По этим причинам мы ограничиваем использование этой библиотеки.  
Но есть исключение — Qt5 объявляет свои алгоритмы устаревшими (deprecated) и сейчас рекомендуется использовать алгоритмы STL (например std::sort вместо qSort).

Заключение

Конечно, может пройти какое-то время, прежде чем Вы сможете использовать новые конструкции С++14 в своих проектах. Но я надеюсь, что вы начнете их применять как и многие другие (Qt Creator, KDE, LLVM). В новых компиляторах MSVC C++14 активен по умолчанию, в clang и gcc нужно использовать специальный флаг (на настоящий момент это -std=c++1y). С помощью qmake можно настроить свой проект на сборку с С++14 начиная с Qt5.4 используя следующую команду: CONFIG += c++14