**ПРАКТИЧЕСКАЯ РАБОТА № 1**

**Тема:** Применение отладочных классов в проекте

**Цель:** изучение свойств и методов встроенных классов отладки проектов и применения этих классов для отладки программ

**Теоретические положения**

Сущность объектно-ориентированного подхода к программированиюзаключается в том, что основные идеи объектно-ориентированного подхода опираются на следующие положения:

- программа представляет собой модель некоторого реального процесса, части реального мира;

- модель реального мира или его части может быть описана как совокупность взаимодействующих между собой объектов;

- объект описывается набором параметров, значения которых определяют состояние объекта, и набором операций (действий), которые может выполнять объект;

- взаимодействие между объектами осуществляется посылкой специальных сообщений от одного объекта к другому. Сообщение, полученное объектом, может потребовать выполнения определенных действий, например, изменения состояния объекта;

- объекты, описанные одним и тем же набором параметров и способные выполнять один и тот же набор действий представляют собой класс однотипных объектов.

Существует ряд техник, предназначенных для обеспечения качества кода, выполняемых по мере его конструирования.

Основные техники обеспечения качества, используемые в процессе конструирования, включают:

1. рефакторинг;

2. модульное (unit) и интеграционное (integration) тестирование;

3. разработка с первичностью тестов (test-first development - тесты пишутся до конструирования кода);

4. пошаговое кодирование (деятельность по конструированию кода разбивается на мелкие шаги, только после тестирования результатов которых производится переход к следующему шагу кодирования; известен также как итеративное кодирование с тестированием);

5. использование процедур утверждений (assertion);

6. отладка (в привычном понимании — debugging);

7. технические обзоры и оценки (review);

8. статический анализ.

Выбор и использование конкретных техник часто диктуется стандартами (внутренними и внешними), используемыми проектной командой, а также зависят от опыта и подготовленности специалистов, занимающихся конструированием кода.

Ключевым аспектом защитного программирования является использование **отладчиков**.

Отладчик (debugger) — программа, предназначенная для поиска ошибок в других программах и позволяет выполнять трассировку, отслеживать, устанавливать или изменять значения переменных в процессе выполнения кода, устанавливать и удалять контрольные точки или условия остановки и т. д.

**Предупреждающие и отладочные сообщения**

Qt включает четыре глобальные функции для записи предупреждающих и отладочного текста. Вы можете использовать их для следующих целей:

[qDebug](http://doc.crossplatform.ru/qt/4.6.x/qtglobal.html#qDebug)() используется для записи пользовательской отладочной информации.

[qWarning](http://doc.crossplatform.ru/qt/4.6.x/qtglobal.html#qWarning)() используется для сообщения предупреждений и устранимых ошибок в вашем приложении.

[qCritical](http://doc.crossplatform.ru/qt/4.6.x/qtglobal.html#qCritical)() используется для записи сообщений о критических ошибках и ошибок системы очетов.

[qFatal](http://doc.crossplatform.ru/qt/4.6.x/qtglobal.html#qFatal)() используется для краткой записи сообщений о фатальной ошибке перед завершением работы.

С помощью этих событий, вы можете разделить ошибки по уровням значимости и применить фильтры, для разделения того, какие ошибки нужно выводить, а какие нет.

Для перенаправления сообщений об ошибках в текстовый файл, вам необходимо установить [CallBack-функцию](https://evileg.com/post/89/) обработчик в приложение. Для этого используется функция **qInstalMessageHandler** .

**Отладка классов в Qt с помощью Qdebug**

Работая с классами в Qt неоднократно приходилось сталкиваться с отладкой, при этом строчка вида:

**qDebug() << value;**

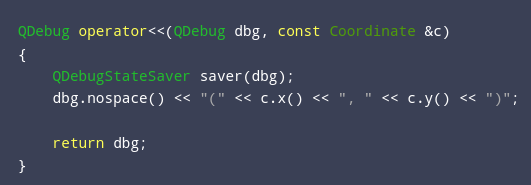
используется очень часто, в том числе, когда нужно контролировать значения некоторых переменных, а запускать отладку долго и не эффективно.

Если требуется вывести значение полей экземпляра класса, то qDebug() ничего не знает о вашем классе и всё, что вы получите в результате - строку вида 0x19e8aa45460.

Можно реализовать оператор потока, используемый qDebug(), чтобы обеспечить поддержку отладки классов.

Класс, реализующий поток, — **QDebug**. Для временного сохранения параметров форматирования потока используется QDebugStateSaver. Манипуляторы nospace() и QTextStream применяются для дальнейшей настройки форматирования.

Пример класса, представляющего двумерную координату.



**Отладка макросов**

Заголовочный файл <QtGlobal> содержит несколько отладочных макросов и директив #define.

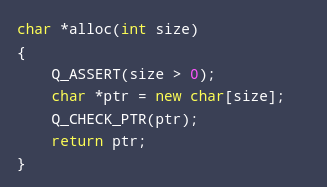
Имеется три важных макроса:

[Q\_ASSERT](http://doc.crossplatform.ru/qt/4.6.x/qtglobal.html#Q_ASSERT)(cond), где cond - булево выражение, записывает предупреждение записывает предупреждение "ASSERT: 'cond' in file xyz.cpp, line 234" и завершает работу, если cond равно false.

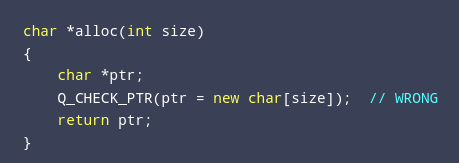
[Q\_ASSERT\_X](http://doc.crossplatform.ru/qt/4.6.x/qtglobal.html#Q_ASSERT_X)(cond, where, what), где cond - булево выражение, where - местоположение, а what - сообщение, записывает предупреждение: "ASSERT failure in where: 'what', file xyz.cpp, line 234" и завершает работу, если cond равно false.

[Q\_CHECK\_PTR](http://doc.crossplatform.ru/qt/4.6.x/qtglobal.html#Q_CHECK_PTR)(ptr), где ptr - указатель. Записывает предупреждение "In file xyz.cpp, line 234: Out of memory" и завершает работу, если ptr равен 0.

Эти макросы используются для обнаружения ошибок программы, например, так:



[Q\_ASSERT](http://doc.crossplatform.ru/qt/4.6.x/qtglobal.html#Q_ASSERT)(), [Q\_ASSERT\_X](http://doc.crossplatform.ru/qt/4.6.x/qtglobal.html#Q_ASSERT_X)() и [Q\_CHECK\_PTR](http://doc.crossplatform.ru/qt/4.6.x/qtglobal.html#Q_CHECK_PTR)() расширяет в ничего, если QT\_NO\_DEBUG определён во время компиляции. По этой причине, аргументы для этих макросов не имеют каких-либо побочных явлений. Вот неправильное использование [Q\_CHECK\_PTR](http://doc.crossplatform.ru/qt/4.6.x/qtglobal.html#Q_CHECK_PTR)():



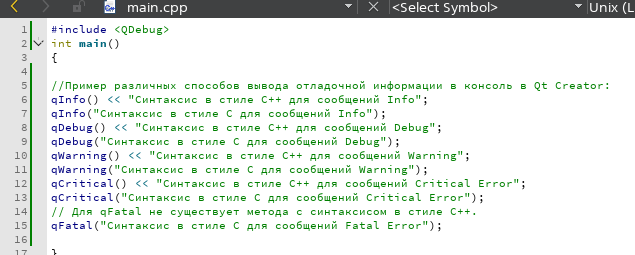
Если этот код скомпилирован с определённым макросом QT\_NO\_DEBUG, код в выражении [Q\_CHECK\_PTR](http://doc.crossplatform.ru/qt/4.6.x/qtglobal.html#Q_CHECK_PTR)() не выполняется и alloc возвращает неинициализированный указатель.

Библиотека Qt содержит сотни внутренних проверок, которые печатают предупреждающие сообщения при обнаружении программной ошибки.

**Технология выполнения:**

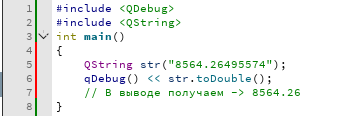
1. Запустить средуQt Creator
2. Выполнить предложенные задания
3. Ответить на контрольные вопросы.

**Задание 1** - Вывод отладочной информации в консоль Qt Creator



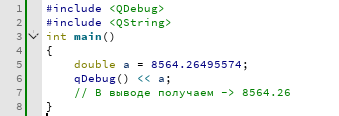
**Задание 2** - Вывод чисел с плавающей запятой через qDebug().

**Постановка задачи:**  имеется объект QString, который содержит некое число "8564.26495574", которое переводим в число с плавающей точкой с помощью метода toDouble(), и проверяем результат с помощью вывода qDebug() , но вывод оказывается ложным.

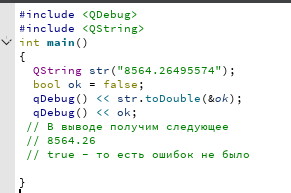


В результате получаем совершенно правильное число, то есть из строки было получено число 8564.26495574 , просто вывод qDebug() показывает округлённый результат.

Тоже самое получим, если выведем значение числа double в qDebug().

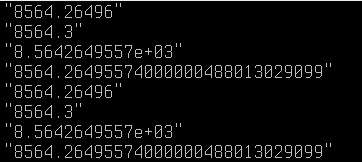
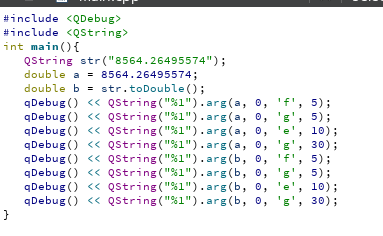


Убедиться в том, что никаких ошибок не происходит при конвертировании числа из QString в double можно, воспользовавшись переменной bool, указатель на которую передаётся в качестве аргумента в метод toDouble().



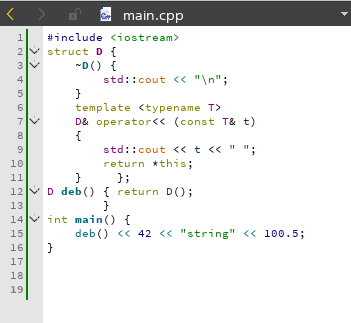
Для того, чтобы сделать корректный вывод, необходимо воспользоваться методом QString::arg() с указанием типа форматирования и точности.

В результате получим следующий вывод qDebug():



**Задание 4 -** Пример использование **operator<< к результату** для класса QDebug

**Постановка задачи:** требуется автоматический перевод курсора вывода на новую строку (т.е. нет явной необходимости добавлять '\n' в конце) через обычный деструктор класса, который вызывается при уничтожении объекта QDebug, который создаётся каждым новым вызовом qDebug().



**Контрольные вопросы**

1. В чем заключается сущность объектно-ориентированного подхода при разработке программного продукта?
2. Какие техники обеспечения качества используются в процессе конструирования?
3. Что такое отладчик?
4. Какие предупреждающие и отладочные сообщения используются в QT?
5. Какие макросы используют при отладке?
6. Для чего используется QDebug?