

Федеральное агентство по образованию  
ГОУ СПО «Южно-Сахалинский Промышленно-экономический техникум»

## **Курсовая работа**

*по предмету «Технология разработки программных продуктов»*

*Тема: «Программа для проведения тестирований»*

Выполнила: студентка гр. ЗП-0701ДТ

Резникова Ольга Леонидовна

Проверил: Никитин Р.В.

Южно-Сахалинск, 2009

## Содержание

Введение. Постановка задачи.....	3
Используемые технологии.....	4
О библиотеке Qt.....	4
О формате XML.....	6
Принцип работы программы.....	11
Структура программы и данных.....	14
Заключение. Перспективы дальнейшего развития программы.....	18
Список литературы.....	19
Приложение: исходные тексты всех классов программы.....	20

## **Введение. Постановка задачи**

Ещё пять-десять лет назад, при почти тотальном господстве операционных систем от Microsoft и не такого уж большого количестве интегрированных сред разработки под них, сами слова «кроссплатформенное программирование» и «свободные технологии» чаще всего порождали вопрос: «А зачем?». Это был замкнутый круг — под PC/Windows было много прикладного ПО, поэтому доля рынка других архитектур и ОС была невелика (а перспективы коммерческого охвата оной были сомнительны), и, соответственно, прикладного ПО под Windows и продолжало выпускаться больше, чем под всё остальное.

Но, поскольку IT — одна из самых быстро развивающихся сфер наукоёмкого производства, ситуация быстро изменяется. Чем дальше — тем больше становится доля Linux-дистрибутивов, используемых на домашних, не серверных, компьютерах. Более того, снова набирают популярность компьютеры Macintosh от компании Apple, последние версии операционной системы для которых также POSIX-совместимы (проще говоря, являют собой вариации на тему Unix). Всё больше и больше разработчиков как коммерческих, так и открытых программных продуктов приходят к выводу, что для популяризации своего детища им так или иначе придётся заставить его работать на разных платформах. И я — не исключение.

Главной целью и задачей данного проекта является написание программы, позволяющей проводить тестирования учащихся по заранее заданным спискам вопросов. Но, в отличие от большинства подобных проектов, пишущихся на Delphi исключительно под Windows и использующих ни с чем не совместимые форматы данных, мною также была поставлена чисто концептуальная задача — использовать самые современные кроссплатформенные технологии проектирования и разработки, а также, по возможности, максимально приблизить написанный код к реально применяемым на производстве промышленным образцам.

Это непростая задача, но, поскольку качественный разработчик программного обеспечения должен максимально «держат нос по ветру» и постоянно следить за состоянием дел в сфере информационных технологий, то это лишь часть того, с чем придётся иметь дело в настоящей и будущей профессиональной деятельности. И именно поэтому я выбрала такую реализацию, казалось бы, совершенно обычного учебного задания.

## **Используемые технологии**

Исходя из поставленной задачи, в моей работе было использовано следующее программное обеспечение:

1. Операционная система OpenSuse Linux 11.1 с рабочей средой KDE 3.5
2. Кроссплатформенная библиотека для создания графических интерфейсов Qt 4.x.x (версии регулярно обновляются)
3. Интегрированная среда разработки KDevelop с интегрированным дизайнером графического интерфейса Qt Designer и справочной системой Qt Assistant
4. Интегрированная среда разработки Qt Creator для сборки под операционную систему Windows
5. Пакет программ для UML-моделирования Visual Paradigm 6.3 (бесплатная версия для некоммерческого использования).
6. В качестве основного языка разработки я выбрала C++ (реализация — открытый компилятор gcc), поскольку это «родной» язык библиотеки Qt. Формат файлов с данными (как с вопросами для теста, так и с результатами его прохождения) — XML.

## **О библиотеке Qt**

Библиотека Qt задумывалась и начиналась как кросс-платформенный тулкит (toolkit) для быстрой разработки графических интерфейсов (GUI) приложений на языке C++, с целью упростить жизнь программистов, пишущих на C++ кросс-платформенные, переносимые GUI-приложения, которые должны работать и в среде Windows, и в среде Unix/Linux под X11, и на компьютерах Macintosh.

В настоящее время Qt значительно переросла рамки тулкита для разработки графических интерфейсов приложений. Она предоставляет использующему её программисту целостный фреймворк (framework), позволяющий при написании большей части приложения использовать только «родные» классы Qt и практически полностью отказаться от написания системно-зависимого кода, использования системных вызовов (будь то Win32 API или Unix system calls) или от изобретения собственных кросс-платформенных обёрток и «велосипедов». Классы Qt покрывают почти все потребности программиста. В Qt

предусмотрены классы и для работы со строками, и для работы с файлами, сетью, базами данных, XML, и для обеспечения многопоточности в приложении, и многое-многое другое. По своим возможностям и богатству библиотека Qt сравнима с .NET Framework или с системой классов Java 2 EE.

Qt предоставляет программисту не только удобный набор библиотек классов, но и определённую модель разработки приложений, определённый каркас их структуры. Следование принципам и правилам «хорошего стиля программирования на C++/Qt» существенно снижает частоту таких трудно отлавливаемых ошибок в приложениях, как утечки памяти (memory leaks), необработанные исключения, незакрытые файлы или неосвобождённые дескрипторы ресурсных объектов, чем нередко страдают программы, написанные «на голем C++» без использования библиотеки Qt.

Важным преимуществом Qt является хорошо продуманный, логичный и стройный набор классов, предоставляющий программисту очень высокий уровень абстракции. Благодаря этому программистам, использующим Qt, приходится писать значительно меньше кода, чем это имеет место при использовании, например, библиотеки классов MFC. Сам же код выглядит стройнее и проще, логичнее и понятнее, чем аналогичный по функциональности код MFC или код, написанный с использованием «родного» для X11 тулкита Xt. Его легче поддерживать и развивать.

Кроме того, даже если программисту в данный конкретный момент не нужна кроссплатформенность для его конкретного приложения (например, планируется версия только для Windows или только для Macintosh), никто не может знать, что понадобится завтра. Бизнес-планы могут поменяться, и может оказаться и нужным, и выгодным выпустить версию для другой операционной системы или другой аппаратной платформы. В случае использования Qt для этого понадобится всего лишь перекомпиляция исходного кода. В случае же использования, например, MFC или «родных» системных API понадобится много тяжёлой работы по портированию, адаптации и отладке, а то и переписыванию с нуля существующего исходного кода для другой ОС или аппаратной платформы.

Многие компании-разработчики приложений Windows используют Qt ещё по одной причине: даже если код пишется и в обозримом будущем будет писаться только для платформы Windows и тестируется только на ней, возможность откомпилировать один и тот же исходный код на одной и той же платформе Windows двумя разными компиляторами (Microsoft Visual C++ и GCC/Win32) гарантирует лучшее качество исходного кода и лучшую

его совместимость со стандартом C++. Что немаловажно для кода, который планируется длительно поддерживать и развивать.[1]

## О формате XML

**XML** (англ. eXtensible Markup Language— расширяемый язык разметки; произносится [экс-эм-эл])— рекомендованный Консорциумом Всемирной паутины язык разметки, фактически представляющий собой свод общих синтаксических правил. XML— текстовый формат, предназначенный для хранения структурированных данных (взамен существующих файлов баз данных), для обмена информацией между программами, а также для создания на его основе более специализированных языков разметки (например, XHTML), иногда называемых словарями. XML является упрощённым подмножеством языка SGML.

Целью создания XML было обеспечение совместимости при передаче структурированных данных между разными системами обработки информации, особенно при передаче таких данных через Интернет. Словари, основанные на XML (например, RDF, RSS, MathML, XHTML, SVG), сами по себе формально описаны, что позволяет программно изменять и проверять документы на основе этих словарей, не зная их семантики, то есть не зная смыслового значения элементов. Важной особенностью XML также является применение так называемых пространств имён (англ. Namespace).

XML — это иерархическая структура, предназначенная для хранения любых данных, визуальную структуру может быть представлена как дерево. Важнейшее обязательное синтаксическое требование — то, что документ имеет только один **корневой элемент** (англ. *root element*) (альтернативно называемый **элементом документа** (англ. *document element*)). Это означает, что текст или другие данные всего документа должны быть расположены между **единственным** начальным корневым тегом и соответствующим ему конечным тегом.

Следующий простейший пример — правильно построенный документ XML:

```
<book>Это книга: "Книжечка"</book>
```

Первая строка XML-документа называется **объявлением XML** (англ. *XML declaration*) — это необязательная строка, указывающая версию стандарта XML (обычно это 1.0), также здесь может быть указана кодировка символов и внешние зависимости.

```
<?xml version="1.0" encoding="UTF-8"?>
```

Спецификация требует, чтобы процессоры XML обязательно поддерживали Юникод-кодировки UTF-8 и UTF-16 (UTF-32 не обязателен). Признаются допустимыми, поддерживаются и широко используются (но не обязательны) другие кодировки, основанные на стандарте ISO/IEC 8859, также допустимы другие кодировки, например, русские Windows-1251, KOI-8.

**Комментарий** может быть размещен в любом месте дерева. XML комментарии размещаются внутри пары тегов `<!--` и `-->`. Два знака дефис (`--`) не могут быть применены ни в какой части внутри комментария.

```
<!-- Это комментарий. -->
```

Ниже приведён пример простого кулинарного рецепта, размеченного с помощью XML:

```
<?xml version="1.0" encoding="UTF-8"?>
<recipe name="хлеб" preptime="5" cooktime="180">
  <title>Простой хлеб</title>
  <ingredient amount="3" unit="стакан">Мука</ingredient>
  <ingredient amount="0.25" unit="грамм">Дрожжи</ingredient>
  <ingredient amount="1.5" unit="стакан">Тёплая вода</ingredient>
  <ingredient amount="1" unit="чайная ложка">Соль</ingredient>
  <Instructions>
    <step>Смешать все ингредиенты и тщательно замесить.</step>
    <step>Закрыть тканью и оставить на один час в тёплом помещении.</step>
    <step>Замесить ещё раз, положить на противень и поставить в
    духовку.</step>
  </Instructions>
</recipe>
```

Остальная часть этого XML-документа состоит из вложенных *элементов*, некоторые из которых имеют *атрибуты* и *содержимое*. **Элемент** обычно состоит из открывающего и закрывающего тегов, обрамляющих текст и другие элементы. **Открывающий тег** состоит из имени элемента в угловых скобках, например, `<<step>>`; **закрывающий тег** состоит из того же имени в угловых скобках, но перед именем ещё добавляется косая черта, например, `<</step>>`. **Содержимым элемента** (англ. *content*) называется всё, что расположено между

открывающим и закрывающим тегами, включая текст и другие (вложенные) элементы. Ниже приведён пример XML-элемента, который содержит открывающий тег, закрывающий тег и содержимое элемента:

```
<step>Замесить ещё раз, положить на противень и поставить в духовку.</step>
```

Кроме содержания у элемента могут быть **атрибуты** — пары имя-значение, добавляемые в открывающий тег после названия элемента. Значения атрибутов всегда заключаются в кавычки (одинарные или двойные), одно и то же имя атрибута не может встречаться дважды в одном элементе. Не рекомендуется использовать разные типы кавычек для значений атрибутов одного тега.

```
<ingredient amount="3" unit="стакан">Мука</ingredient>
```

В приведённом примере у элемента «`ingredient`» есть два атрибута: «`amount`», имеющий значение «3», и «`unit`», имеющий значение «стакан». С точки зрения XML-разметки, приведённые атрибуты не несут никакого смысла, а являются просто набором символов.

Кроме текста, элемент может содержать другие элементы:

```
<Instructions>
  <step>Смешать все ингредиенты и тщательно замесить.</step>
  <step>Закрыть тканью и оставить на один час в тёплом помещении.</step>
  <step>Замесить ещё раз, положить на противень и поставить в
духовку.</step>
</Instructions>
```

В данном случае элемент «`Instructions`» содержит три элемента «`step`». XML не допускает перекрывающихся элементов. Например, приведённый ниже фрагмент некорректен, так как элементы «`em`» и «`strong`» перекрываются.

```
<!-- ВНИМАНИЕ! Некорректный XML! -->
<p>Обычный <em>акцентированный <strong>выделенный и акцентированный</em>
выделенный</strong></p>
```

Каждый XML-документ должен содержать в точности один **корневой элемент** ([англ. \*root element\*](#) или *document element*), таким образом, следующий фрагмент не может считаться

корректным XML-документом.

```
<!-- ВНИМАНИЕ! Некорректный XML! -->
<thing>Сущность №1</thing>
<thing>Сущность №2</thing>
```

Для обозначения элемента без содержания, называемого **пустым элементом**, необходимо применять особую форму записи, состоящую из одного тега, в котором после имени элемента ставится косая черта. Если в DTD элемент не объявлен пустым, но в документе он не имеет содержания, для него *допускается* применять такую форму записи. Например:

```
<foo></foo>
<foo />
<foo/>
```

В XML определены два метода записи специальных символов: ссылка на сущность и ссылка по номеру символа. **Сущностью** (англ. *entity*) в XML называются именованные данные, обычно текстовые, в частности спецсимволы. **Ссылка на сущность** (англ. *entity references*) указывается в том месте, где должна быть сущность и состоит из амперсанда («&»), имени сущности и точки с запятой («;»). В XML есть несколько predefined сущностей, таких как «lt» (ссылаться на неё можно написав «&lt;») для левой угловой скобки и «amp» (ссылка — «&amp;») для амперсанда, возможно также определять собственные сущности. Помимо записи с помощью сущностей отдельных символов, их можно использовать для записи часто встречающихся текстовых блоков. Ниже приведён пример использования predefined сущности для избежания использования знака амперсанда в названии:

```
<company-name>AT&amp;T</company-name>
```

Полный список predefined сущностей состоит из &amp; («&»), &lt; («<»), &gt; («>»), &apos; («'»), и &quot; («"») — последние две полезны для записи разделителей внутри значений атрибутов. Определить свои сущности можно в DTD-документе.

Иногда бывает необходимо определить неразрывный пробел, который очень часто используется в HTML и обозначается как &nbsp; в XML такой predefined сущности нет, его записывают &#160, а использование &nbsp; вызывает ошибку. Отсутствие этой

весьма распространённой сущности у множества программистов, зачастую, вызывает удивление, и это создаёт некоторые трудности при миграции своих HTML-разработок в XML;

**Ссылка по номеру символа** (англ. *numeric character reference*) выглядит как ссылка на сущность, но вместо имени сущности указывается символ # и число (в десятичной или шестнадцатеричной записи), являющееся номером символа в кодовой таблице Юникод. Это обычно символы, которые невозможно закодировать напрямую, например буква арабского алфавита в ASCII-кодированном документе. Амперсанд может быть представлен следующим образом:

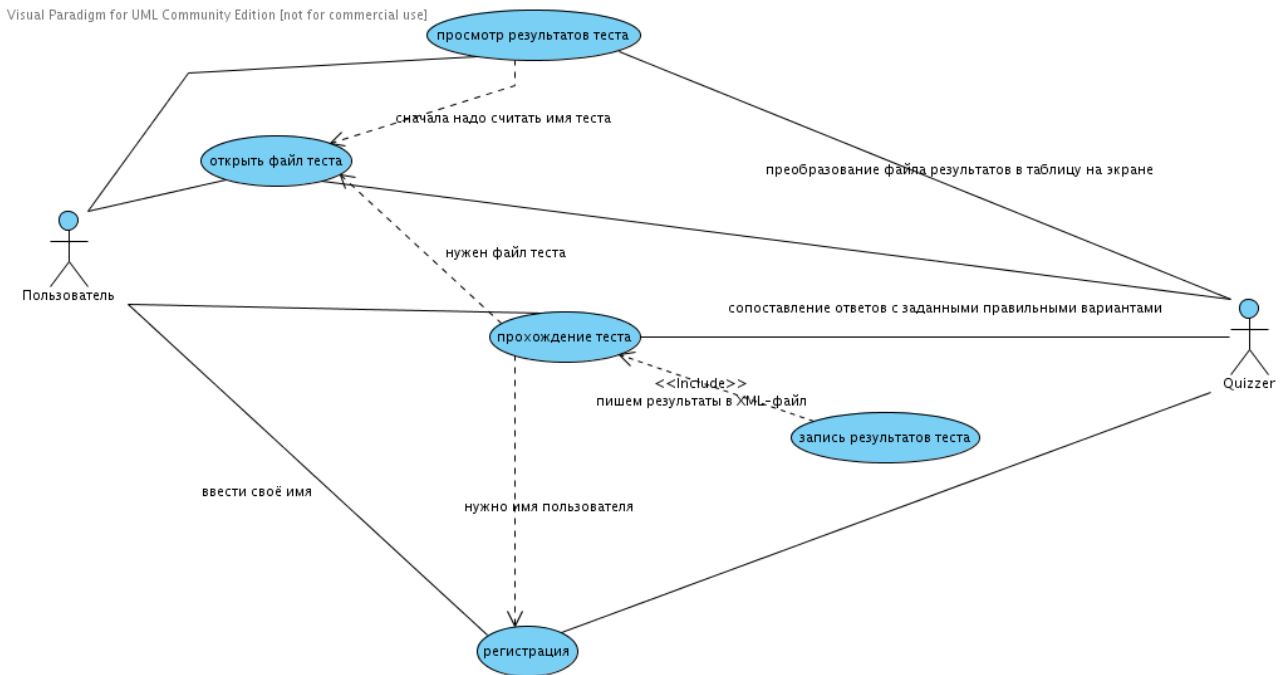
```
<company-name>AT&#38;T</company-name>
```

Существует ещё множество правил, касающихся составления корректного XML-документа, но целью данного краткого обзора было лишь показать основы, необходимые для понимания структуры XML-документа.[2]

На этом я закончу теоретическую часть своей работы и перейду к практической — пришла пора посмотреть на программу как в действии, так и изнутри.

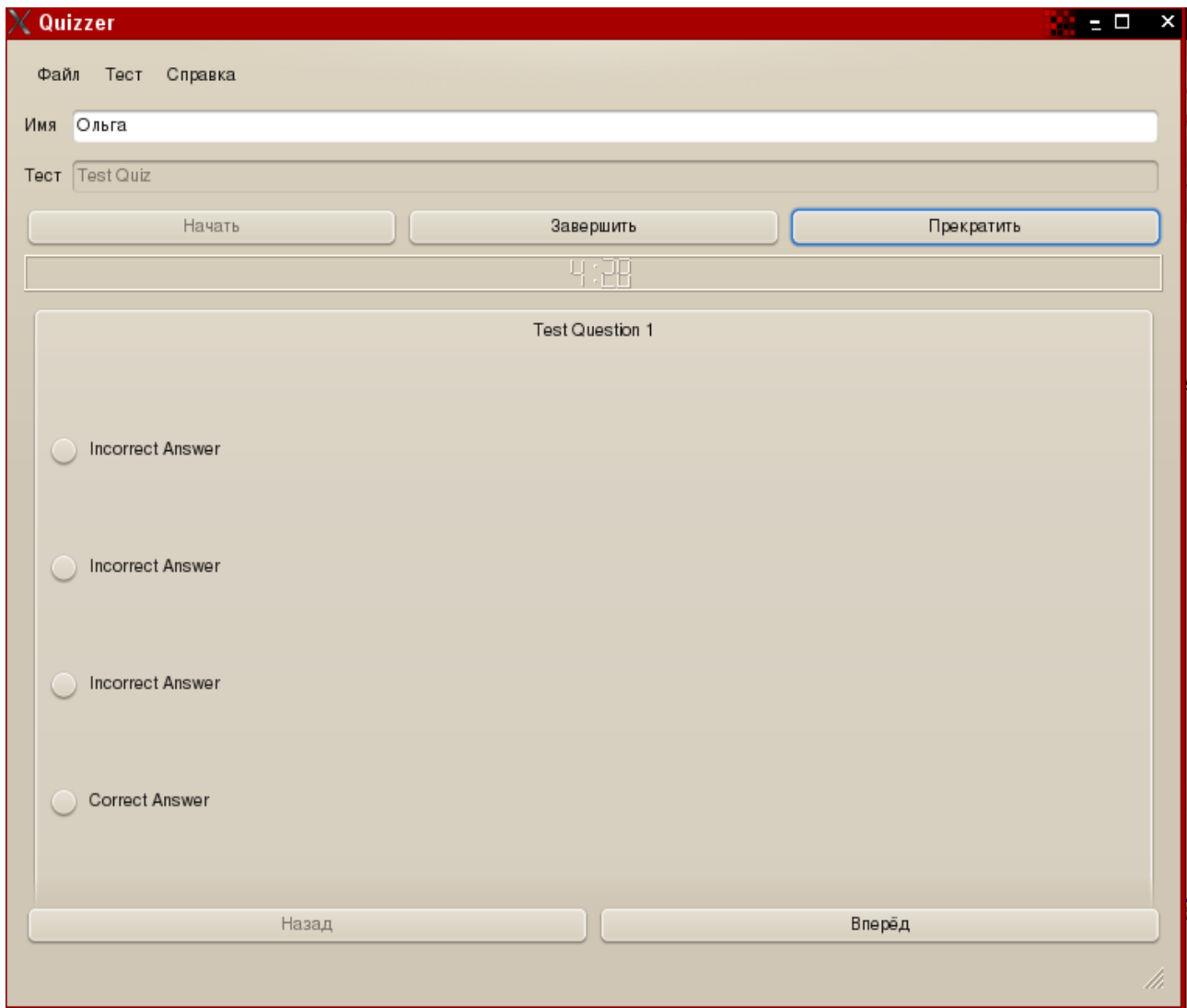
## Принцип работы программы

Моя программа, в честь библиотеки Qt и тамошней традиции начинать имена всех классов с «Q», получила название Quizzer (от англ. Quiz - «вопросник, тест»). Функционал программы, её механизм взаимодействия с пользователем отражает следующая диаграмма вариантов использования стандарта UML:



По этой диаграмме видно, что первоочерёдными действиями пользователя являются открытие файла с тестом и регистрация (т.е., ввод своего имени в специально отведённое для этого поле), после чего становятся доступными прохождение теста (представляющее собой сравнение выбранных ответов с заданными и включающее в себя запись полученного результата в файл) и просмотр ранее набранных пользователями результатов. Файлы с тестами по умолчанию размещаются в папке **quizzes** дистрибутива программы (впрочем, можно открывать и файлы из других папок), файлы с результатами же создаются автоматически в папке **results** (имя файла с результатами — такое же, как и название теста).

Ниже представлена главная форма программы в действии (так она выглядит в рабочей среде KDE):



На ней расположены основные управляющие элементы: меню, текстовые поля для имени пользователя и названия теста, кнопки управления прохождением теста («начать», «завершить» и «прекратить»), индикатор оставшегося времени, стопка вопросов с вариантами ответов в виде радио-кнопок, а также кнопки выбора текущего вопроса («назад» и «вперёд»).

У приложения есть и вторая форма — форма результатов. Это модальное диалоговое окно, содержащее табличный виджет, где представлены количество попыток и лучший (по количеству правильно отвеченных вопросов) результат каждого пользователя. По каждому из столбцов таблицы возможна сортировка.

	Пользователь ^	Попыток	Лучший результат
1	ывавыва	1	0
2	dfgdfgdg	1	3
3	Olga	2	3
4	Ad_Astra	4	5

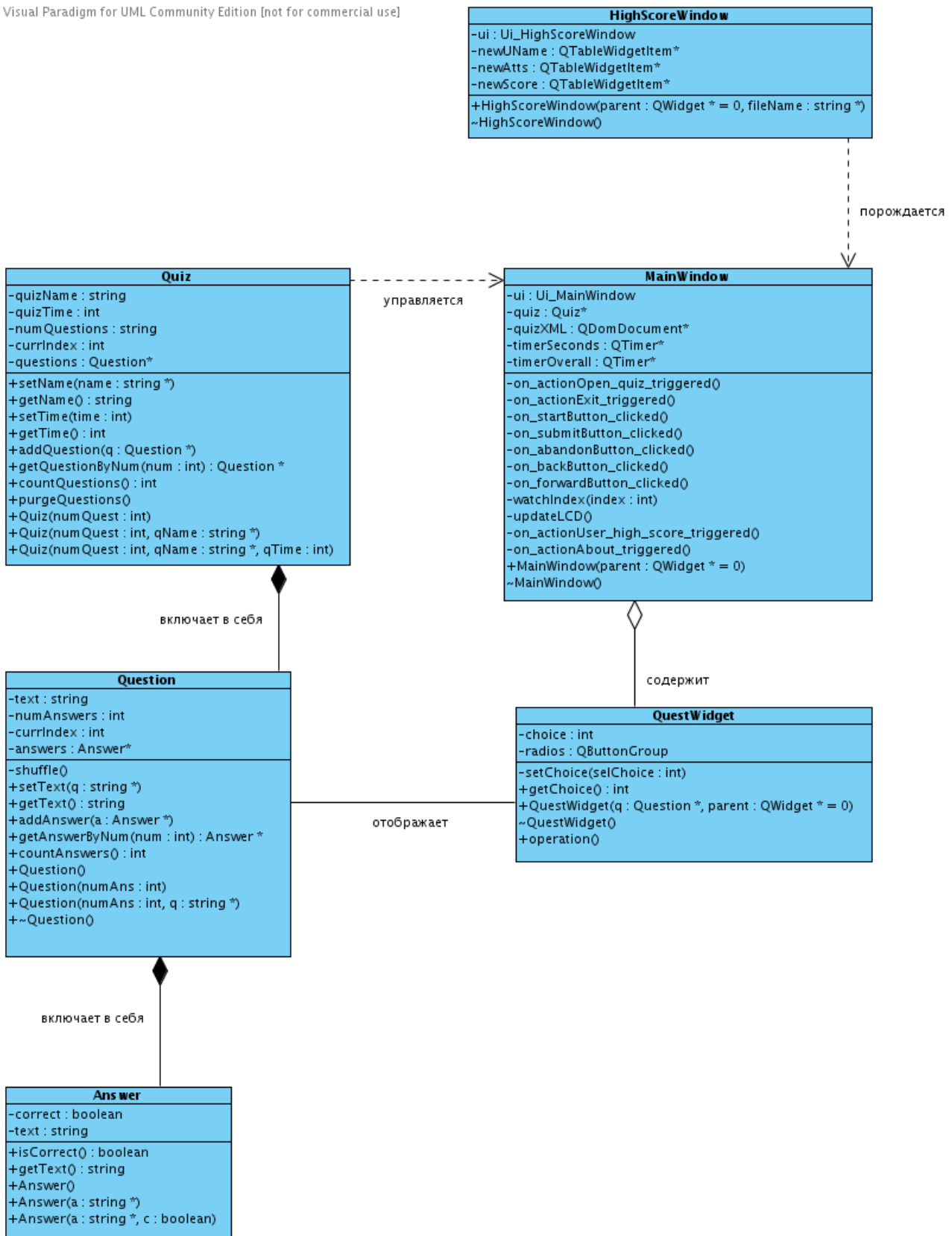
Обе формы физически являют собой файлы того же формата XML, где декларативно описаны основные объекты, имеющиеся на формах (все файлы есть на прилагающемся к работе диске).

## **Структура программы и данных**

В общем и целом, Quizzer написан в типичной объектно-ориентированной стилистике C++. Программа состоит из классов (разбитых на заголовочный файл и файл реализации для удобства подключения), от которых образуются объекты. Между этими объектами и идёт взаимодействие в коде программы.

Каждый класс несёт в себе набор свойств и методов поведения, определяющих его назначение и использование. Также классы могут содержать конструкторы и деструкторы — особые методы, вызываемые при создании объекта или его разрушении, и инициализирующие или удаляющие динамически генерируемые объектом ресурсы во избежание утечек памяти (к сожалению, в стандарте C++ не предусмотрено автоматического «сборщика мусора», и память при удалении динамически созданных переменных приходится освобождать вручную).

Список написанных мной классов и взаимоотношения между ними отражены на следующей диаграмме классов стандарта UML:



По этой диаграмме мы видим, что главный управляющий класс приложения — это класс `MainWindow`, отвечающий за основную форму. Он содержит обработчики основных событий формы (нажатия управляющих кнопок или выбора пунктов меню). Главная форма может вызывать объект класса `HighScoreWindow` — диалоговое окно с лучшими результатами (единственное, что это окно умеет делать — читать информацию из соответствующего файла и выводить её в удобную таблицу).

Также главная форма одним из своих атрибутов содержит указатель на объект класса `Quiz`, то есть управляет тестом, параметры которого считываются из соответствующего файла. Форма может считать тест из файла, запустить тест, завершить его и записать в файл полученные результаты, или же прервать и очистить все параметры.

Тест состоит из вопросов, а вопросы — из текста и вариантов ответов, среди которых есть один правильный. На диаграмме классов это отражено через отношение агрегации (включения), в коде программы каждому классу назначены соответствующие методы для получения его элементов, а формат файлов теста, понимаемый программой, выглядит так:

```
<?xml version="1.0"?>
<quiz name="название_теста" time="время_теста_в_секундах">
  {
    <question name="текст_вопроса">
      <answer correct="true" text="правильный_ответ" />
      {
        <answer text="неправильный_ответ"/>
        <answer text="неправильный_ответ"/>
        <answer text="неправильный_ответ"/>
      } 1-... раз
    } 1-... раз
  </question>
</quiz>
```

По этому шаблону можно составить файл теста на любую тематику. Следует отдельно отметить, что количество вопросов и ответов может быть произвольным (в разумных пределах), так как массивы, отвечающие за них, инициализируются динамически, их размерность не известна заранее. Варианты ответов перетасовываются в ходе работы программы (чтобы при повторном прохождении теста не было механического запоминания вариантов), порядок вопросов же остаётся неизменным (так как тесты могут содержать логическую последовательность из вопросов).

Также главная форма содержит виджет-стопку `QStackedWidget`, фактически являющийся контейнером для других виджетов, среди которых в каждый момент времени является активным и видимым только один, а остальные как бы лежат стопкой под ним. Это подходит для нашей цели, и в нашем случае в стопке размещаются виджеты класса `QuestWidget` — это унаследованные от системного класса `QWidget` комбинации рамки с

заголовком (где размещается текст вопроса) и группы радиокнопок с вариантами ответов, способных «запоминать» текущий выбор через метод-обработчик соответствующего события.

Эти запомненные варианты при нажатии кнопки «завершить» или исходе времени и сравниваются с прочитанными из файла теста правильными вариантами. После чего результат пишется в файл, имеющий следующую структуру:

```
<?xml version='1.0'?>
<results>
  <user name="имя_пользователя" >
    <result percentage="результат_в_%" score="число_правильных_ответов" />
    <result percentage="результат_в_%" score="число_правильных_ответов" />
    <result percentage="результат_в_%" score="число_правильных_ответов" />
  </user>
</results>
```

Пользователей и результатов для каждого из них, соответственно, может также быть неограниченное количество (а чтобы не засорять таблицу результатов при вызове соответствующего окна, в ней отображается лучший из них).

Такова внутренняя логика моей программы. Разумеется, при желании можно её дополнить и улучшить (об этом — далее), но основная задача курсовой работы уже реализована.

## ***Заключение. Перспективы дальнейшего развития программы***

Конечно, это всего лишь учебный проект, очень сильно ограниченный форматом курсовой работы. Но чтобы и дальше приблизить его к реально используемым на производстве паттернам разработки ПО, есть много областей для улучшения.

Например, можно добавить в программу валидацию XML-файлов с вопросами и результатами по описательным файлам DTD или XML Schema (в том варианте, который используется сейчас, возможны ошибки из-за чтения XML-файла другой, не отражающей структуры теста, семантики). Если добавить такую валидацию и в случае её непрохождения вызвать исключение — то программа не станет и пробовать читать структуру такого файла, что сэкономит время и нервы пользователя.

Также можно и пойти дальше по пути разделения бизнес-логики и представления, и организовать нечто вроде «ядра», промежуточного слоя между объектами формы и логикой работы приложения. В больших проектах это часто становится необходимостью, так как позволяет легко менять объекты, представляющие данные, оставляя неизменными вызовы всех остальных функций. Другое дело, что этот проект совсем небольшой, и такое разделение тут будет скорее излишним усложнением структуры, нежели приданием ей гибкости.

Также, проявив немного фантазии, можно организовать это приложение как клиент-серверное и вести централизованную базу пользователей и тестов.

Всё это изрядно выходит за рамки курсовой работы, да — но мной принимается во внимание и, возможно, рано или поздно, просто в учебно-исследовательских целях, будет в программу добавлено. Пока же я остановлюсь на имеющемся варианте, сочтя основную задачу — написание кроссплатформенной программы по стандартному техническому заданию — выполненной. Исходный код всех классов можно посмотреть в приложении к данной работе, рабочий же вариант (включающий также файлы форм и прочие служебные файлы) прилагаю к работе в виде файлов на отдельном носителе.

## **Список литературы**

1: Викиучебник, "Программирование на C++ с использованием Qt", 2008-2009 -

[http://ru.wikibooks.org/wiki/%D0%9F%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5\\_%D0%BD%D0%B0\\_C%2B%2B\\_%D1%81\\_%D0%B8%D1%81%D0%BF%D0%BE%D0%BB%D1%8C%D0%B7%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5%D0%BC\\_%D0%B1%D0%B8%D0%B1%D0%BB%D0%B8%D0%BE%D1%82%D0%B5%D0%BA%D0%B8\\_Qt](http://ru.wikibooks.org/wiki/%D0%9F%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5_%D0%BD%D0%B0_C%2B%2B_%D1%81_%D0%B8%D1%81%D0%BF%D0%BE%D0%BB%D1%8C%D0%B7%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5%D0%BC_%D0%B1%D0%B8%D0%B1%D0%BB%D0%B8%D0%BE%D1%82%D0%B5%D0%BA%D0%B8_Qt)

2: Wikipedia, "XML", 2004-2009 - <http://ru.wikipedia.org/wiki/XML>

## **Приложение: исходные тексты всех классов программы**

```
#include <QApplication>
#include "mainform.h"

/**
 * главная функция приложения (точка входа)
 * @param argc
 * @param argv[]
 * @return статус завершения приложения
 */
int main(int argc, char *argv[])
{
    QApplication app(argc, argv);
    MainWindow window;
    window.show();
    return app.exec();
}

//
// C++ Interface: mainform
//
// Description: Заголовок класса главной формы
//
//
// Author: Ad_Astra <ad-astra@ad-astra.name>, (C) 2008
//
// Copyright: See COPYING file that comes with this distribution
//
//
#ifndef QUIZZER_H
#define QUIZZER_H
#include <QtXml>
#include "ui_quizzer.h"
#include "quiz.h"

class MainWindow : public QWidget
{
    Q_OBJECT

public:
    MainWindow(QWidget *parent = 0);
    ~MainWindow();

private slots:
    void on_actionOpen_quiz_triggered();
    void on_actionExit_triggered();
    void on_startButton_clicked();
    void on_submitButton_clicked();
    void on_abandonButton_clicked();
    void on_backButton_clicked();
    void on_forwardButton_clicked();
    void watchIndex(int index);
    void updateLCD();
    void on_actionUser_high_score_triggered();
    void on_actionAbout_triggered();

private:
    Ui_MainWindow ui;
```

```

    Quiz *quiz;
    QDomDocument *quizXML;
    QTimer *timerSeconds;
    QTimer *timerOverall;
};

#endif

//
// C++ Implementation: mainform
//
// Description: Основной класс формы, управляющий реакциями на нажатия,
// таймерами и т.п.
//
// Author: Ad_Astra <ad-astra@ad-astra.name>, (C) 2008
//
// Copyright: See COPYING file that comes with this distribution
//
//
#include <QtGui>
#include <QCoreApplication>
#include <QtXml>
#include "mainform.h"
#include "highscore.h"
#include "quiz.h"
#include "questwidget.h"

/**
 * Это конструктор формы. В нём присваиваются начальные значения используемым
 * переменным, а также соединяются между собой сигналы (происходящие события) и
 * слоты (методы, специально предназначенные для их обработки)
 * @param parent
 */
MainWindow::MainWindow(QWidget *parent)
    : QWidget(parent), quizXML(new QDomDocument), quiz(0)
{
    ui.setupUi(this);
    connect(ui.questionsStack, SIGNAL(currentChanged(int)), this,
SLOT(watchIndex(int)));
    timerSeconds = new QTimer(this);
    timerSeconds->setSingleShot(false);
    connect(timerSeconds, SIGNAL(timeout()), this, SLOT(updateLCD()));
    timerOverall = new QTimer(this);
    timerOverall->setSingleShot(true);
    connect(timerOverall, SIGNAL(timeout()), this,
SLOT(on_submitButton_clicked()));
}

/**
 * Деструктор класса, освобождающий объекты формы при её закрытии,
 * во избежание утечек памяти
 */
MainWindow::~MainWindow() {
    delete timerSeconds;
    delete timerOverall;
    delete quiz;
    delete quizXML;
}

/**
 * Эта функция открывает и обрабатывает XML-файл с тестом, создавая объект

```

```

* класса Quiz и устанавливая соответствующие значения главному таймеру и полю
* названия теста
* @param checked
*/
void MainWindow::on_actionOpen_quiz_triggered() {

    QString quizFileName = QFileDialog::getOpenFileName(this,
QString::fromUtf8("Открыть файл"), "./quizzes", tr("XML Files (*.xml)"));
    QFile *quizFile = new QFile(quizFileName);
    try {
        if (!quizFile->open(QIODevice::ReadOnly)) throw("Не могу открыть
файл!");
        if (!quizXML->setContent(quizFile)) throw("Это не XML-файл!");
        ui.startButton->setEnabled(true);
        ui.abandonButton->setEnabled(true);
        ui.actionStart->setEnabled(true);
        ui.actionAbandon->setEnabled(true);
        ui.actionUser_high_score->setEnabled(true);
        QDomElement rootElement;
        rootElement = quizXML->documentElement();
        QDomNodeList questionNodes = quizXML->elementsByTagName("question");
        int numQuestions = questionNodes.length();
        quiz = new Quiz(numQuestions, &rootElement.attribute("name"),
rootElement.attribute("time").toInt());
        ui.QuizName->setText(quiz->getName());
        ui.lcdNumber->display(QString::number(quiz-
>getTime()/60)+":"+QString::number(quiz->getTime()%60));
        quizFile->close();
        delete quizFile;
    }
    catch (char const* errName) {
        QMessageBox::warning(0, "Error", errName);
    }
}

/**
* Эта функция закрывает приложение при выборе пункта меню "Выход"
* @param checked
*/
void MainWindow::on_actionExit_triggered() {
    QCoreApplication::quit();
}

/**
* Эта функция запускает тест при нажатии соответствующей кнопки или выборе
* пункта меню. Здесь из файла читаются вопросы и ответы (для каждого из них
* создаётся объект класса, соответственно, Question или Answer), и "на лету"
* генерируются графические виджеты для выбора ответа.
*/
void MainWindow::on_startButton_clicked() {
    try {
        if(ui.userName->text()=="") throw "Вы должны ввести своё имя!";
        //заново устанавливаем время, на случай, если этот тест уже был
        пройден
        QDomElement rootElement = quizXML->documentElement();
        quiz->setTime(rootElement.attribute("time").toInt());
        QDomNodeList questionNodes = quizXML->elementsByTagName("question");
        int numQuestions = quiz->countQuestions();
        //если остались видимыми вопросы от ранее пройденного теста -
        удаляем их
        while (ui.questionsStack->count() > 0) {

```

```

        QWidget *qw = ui.questionsStack->currentWidget();
        ui.questionsStack->removeWidget(qw);
        delete(qw);
    }
    //формируем вопросы и варианты ответов
    for (uint i = 0; i<numQuestions; i++) {
        QDomNodeList ans = questionNodes.item(i).childNodes();
        uint numAns = ans.length();
        const QString qName =
questionNodes.item(i).toElement().attribute("name");
        Question *quest;
        quest = new Question(numAns, &qName);
        for(uint j = 0; j<numAns; j++) {
            QString aName =
ans.item(j).toElement().attribute("text");
            Answer *a;
            if (ans.item(j).toElement().hasAttribute("correct") and
ans.item(j).toElement().attribute("correct")=="true") {
                a = new Answer(&aName, true);
            }
            else {
                a = new Answer(&aName);
            }
            quest->addAnswer(a);
        }
        quiz->addQuestion(quest);
        QuestWidget *qw = new QuestWidget(quest);
        ui.questionsStack->addWidget(qw);
    }
    ui.questionsStack->setCurrentIndex(0);
    ui.startButton->setEnabled(false);
    ui.submitButton->setEnabled(true);
    ui.abandonButton->setEnabled(true);
    ui.actionStart->setEnabled(false);
    ui.actionSubmit->setEnabled(true);
    ui.actionAbandon->setEnabled(true);
    ui.forwardButton->setEnabled(true);
    ui.lcdNumber->display(QString::number(quiz-
>getTime()/60)+" "+QString::number(quiz->getTime()%60));
    timerSeconds->start(1000);
    timerOverall->start(quiz->getTime()*1000);
}
catch (char const* errName) {
    QMessageBox::warning(0, "Error", QString::fromUtf8(errName));
}
}

/**
 * Эта функция вызывается при выборе вопроса, и проверяет, не первый ли он
 * и не последний ли он, чтобы корректно выводить кнопки "вперёд" и "назад"
 * @param index
 */
void MainWindow::watchIndex(int index) {
    const int lastIndex = ui.questionsStack->count()-1;
    if (index==0) {
        ui.backButton->setEnabled(false);
        ui.forwardButton->setEnabled(true);
    } else if(index == lastIndex) {
        ui.backButton->setEnabled(true);
        ui.forwardButton->setEnabled(false);
    }
}

```

```

        else {
            ui.backButton->setEnabled(true);
            ui.forwardButton->setEnabled(true);
        }
    }

/**
 * Эта функция обновляет содержимое LCD-индикатора при отсчёте секунд
 */
void MainWindow::updateLCD() {
    quiz->setTime(quiz->getTime()-1);
    QString timeDisplay = QString::number(quiz-
>getTime()/60)+"."+QString::number(quiz->getTime()%60);
    ui.lcdNumber->display(timeDisplay);
}

/**
 * Эта функция делает активным предыдущий вопрос при нажатии кнопки "назад"
 */
void MainWindow::on_backButton_clicked() {
    ui.questionsStack->setCurrentIndex(ui.questionsStack->currentIndex()-1);
}

/**
 * Эта функция делает активным следующий вопрос при нажатии кнопки "вперёд"
 */
void MainWindow::on_forwardButton_clicked() {
    ui.questionsStack->setCurrentIndex(ui.questionsStack->currentIndex()+1);
}

/**
 * Эта функция вызывается при нажатии кнопки "завершить" или выборе
 * соответствующего пункта меню. Она обрабатывает заполненные ответы, вычисляет
 * количество правильных и пишет результат в файл.
 */
void MainWindow::on_submitButton_clicked() {
    //подсчитываем количество правильных ответов
    int correctAnswers = 0;
    int choice = -1;
    for(int cnt=0; cnt<ui.questionsStack->count(); cnt++) {
        QuestWidget *currWgt = (QuestWidget*) ui.questionsStack-
>widget(cnt);
        choice = currWgt->getChoice();
        if (choice > -1) {
            if (quiz->getQuestionByNum(cnt)->getAnswerByNum(choice)-
>isCorrect()==true) {
                correctAnswers++;
            }
        }
    }
    QString corr;
    corr.setNum(correctAnswers);
    QString all;
    all.setNum(ui.questionsStack->count());
    //пишем всё в XML-файл, если он существует... а если не существует,
создаём и всё равно пишем
    QFile *resultsFile = new QFile("./results/" + quiz->getName() + ".xml" );
    resultsFile->open(QIODevice::ReadOnly);
    QDomDocument *resultsXML = new QDomDocument();
    QDomElement resRoot;
    if (!resultsXML->setContent(resultsFile) ) {

```

```

        resRoot = resultsXML->createElement("results");
        resultsXML->appendChild(resRoot);
    } else {
        resRoot = resultsXML->documentElement();
    }
    QDomNodeList userList = resultsXML->elementsByTagName("user");
    QDomElement user;

    if (userList.count() > 0) {
        for (int cnt = 0; cnt < userList.count(); cnt++) {

            if (userList.item(cnt).toElement().attribute("name") == ui.userName->text())
            {
                user = userList.item(cnt).toElement();
            }
        }
    }
    if (user.isNull()) {
        user = resultsXML->createElement("user");
        user.setAttribute("name", ui.userName->text());
    }
    resultsFile->close();
    QDomElement result = resultsXML->createElement("result");
    result.setAttribute("score", corr);
    QString percentage = QString::number(((double)correctAnswers)/quiz-
>countQuestions()) * 100);
    result.setAttribute("percentage", percentage);
    user.appendChild(result);
    resRoot.appendChild(user);

    resultsFile->open(QFile::WriteOnly | QFile::Truncate);
    QTextStream out(resultsFile);
    out << resultsXML->toString();
    delete resultsFile;
    //выводим информацию и ставим виджеты в начальное положение
    QMessageBox::information(0, QString::fromUtf8("Результат"),
corr+QString::fromUtf8(" правильных ответов из ") + all + (" +percentage+%"));
    ui.actionStart->setEnabled(true);
    ui.actionSubmit->setEnabled(false);
    ui.actionAbandon->setEnabled(false);
    ui.startButton->setEnabled(true);
    ui.submitButton->setEnabled(false);
    ui.abandonButton->setEnabled(false);
    quiz->setTime(0);
    quiz->purgeQuestions();
    ui.lcdNumber->display(0);
    timerSeconds->stop();
}

/**
 * Эта функция вызывается при нажатии кнопки "Прервать" и очищает все
 * заполненные значения
 */
void MainWindow::on_abandonButton_clicked() {
    QMessageBox msgBox;
    msgBox.setText(QString::fromUtf8("Вы действительно хотите прекратить?"));
    msgBox.setInformativeText(QString::fromUtf8("Все ваши ответы будут
потеряны"));
    msgBox.setStandardButtons(QMessageBox::Cancel | QMessageBox::Ok);
    msgBox.setDefaultButton(QMessageBox::Cancel);
    int ret = msgBox.exec();
}

```

```

if (ret==QMessageBox::Ok) {
    QString *nullName= new QString;
    quiz->setName(nullName);
    quiz->setTime(0);
    quiz->purgeQuestions();
    ui.QuizName->setText("");
    ui.lcdNumber->display(0);
    timerSeconds->stop();
    while (ui.questionsStack->count() > 0) {
        QWidget *qw = ui.questionsStack->currentWidget();
        ui.questionsStack->removeWidget(qw);
        delete(qw);
    }
    ui.backButton->setEnabled(false);
    ui.forwardButton->setEnabled(false);
    ui.startButton->setEnabled(false);
    ui.submitButton->setEnabled(false);
    ui.abandonButton->setEnabled(false);
    ui.actionStart->setEnabled(false);
    ui.actionSubmit->setEnabled(false);
    ui.actionAbandon->setEnabled(false);
    ui.actionUser_high_score->setEnabled(false);
}
}

/**
 * Эта функция вызывается при выборе пункта меню "Результаты" и выводит
 * модальное окно
 * с результатами, которых пользователи достигли при прохождении этого теста
 */
void MainWindow::on_actionUser_high_score_triggered() {
    HighScoreWindow* hsWindow = new HighScoreWindow(this, &(quiz->getName()));
    hsWindow->show();
}

void MainWindow::on_actionAbout_triggered() {
    QMessageBox::information(0, QString::fromUtf8("Quizzer v 1.0"),
    QString::fromUtf8("Курсовая работа Резниковой О.Л. а.к.а. Ad_Astra"));
}

//
// C++ Interface: quiz
//
// Description: Заголовок класса для теста
//
//
// Author: Ad_Astra <ad-astra@ad-astra.name>, (C) 2009
//
// Copyright: See COPYING file that comes with this distribution
//
//
#ifndef QUIZ_H
#define QUIZ_H
#include <QCoreApplication>
#include "question.h"
class Quiz {
public:
    Quiz(const uint numQuest);
    Quiz(const uint numQuest, const QString *qName);
    Quiz(const uint numQuest, const QString *qName, const int qTime);
    ~Quiz();

```

```

    void setName(const QString *qName);
    QString getName();
    void setTime(const int qTime);
    int getTime();
    void addQuestion(Question *q);
    Question *getQuestionByNum(const uint num);
    uint countQuestions();
    void purgeQuestions();

private:
    QString name;
    int time;
    uint numQuestions;
    Question *questions;
    int currIndex;
};
#endif

//
// C++ Implementation: quiz
//
// Description: Класс для теста, связанного с главной формой и генерируемого из
// файла
//
// Author: Ad_Astra <ad-astra@ad-astra.name>, (C) 2009
//
// Copyright: See COPYING file that comes with this distribution
//
//
#include <QCoreApplication>
#include "quiz.h"
#include "question.h"
/**
 * Базовый конструктор класса, создающий тест с пустыми названием и временем,
 * но с заданным количеством вопросов
 * @param numQuest количество вопросов
 */
Quiz::Quiz(const uint numQuest) : name(""), time(0), numQuestions(numQuest),
questions(new Question[numQuest]), currIndex(0) {}
/**
 * Этот конструктор создаёт тест с заданными названием и кол-вом вопросов,
 * время же устанавливает в 0, и его придётся добавить позднее
 * @param numQuest количество вопросов
 * @param name строка типа QString
 */
Quiz::Quiz(const uint numQuest, const QString *qName) : name(*qName), time(0),
numQuestions(numQuest), questions(new Question[numQuest]), currIndex(0) {}
/**
 * Наиболее параметризованный конструктор класса, позволяющий создать тест по
 * названию, заданному количеству вопросов и времени
 * @param numQuest количество вопросов
 * @param name строка типа QString
 * @param time время в секундах
 */
Quiz::Quiz(const uint numQuest, const QString *qName, int qTime) : name(*qName),
time(qTime), numQuestions(numQuest), questions(new Question[numQuest]),
currIndex(0) {}
/**
 * Деструктор класса очищает список вопросов при удалении теста
 */
Quiz::~Quiz() {

```

```

        delete[] questions;
    }

/**
 * Эта функция присваивает объекту теста имя
 * @param name указатель на строку типа QString
 */
void Quiz::setName(const QString *qName) {
    name = *qName;
}

/**
 * Эта функция получает имя теста
 * @return строка типа QString
 */
QString Quiz::getName() {
    return name;
}

/**
 * Эта функция задаёт объекту теста время на его выполнение
 * @param time время в секундах
 */
void Quiz::setTime(const int qTime) {
    time = qTime;
}

/**
 * Эта функция получает время, отведённое на выполнение теста
 * @return время в секундах
 */
int Quiz::getTime() {
    return time;
}

/**
 * Эта функция добавляет ответ к имеющемуся массиву ответов
 * @param q объект класса Question
 */
void Quiz::addQuestion(Question *q) {
    questions[currIndex] = *q;
    currIndex++;
}

/**
 * Эта функция определяет вопрос, идущий под заданным номером в списке вопросов,
 * принадлежащих данному тесту
 * @param num номер вопроса
 * @return указатель на объект класса Question
 */
Question *Quiz::getQuestionByNum(const uint num) {
    return &questions[num];
}

/**
 * Эта функция - акцессор к счётчику вопросов, собственно, выводит значение
 * счётчика
 * @return число вопросов
 */
uint Quiz::countQuestions() {
    return numQuestions;
}

```

```

}

/**
 * Эта функция очищает параметры объекта без удаления самого объекта
 */
void Quiz::purgeQuestions() {
    currIndex = 0;
    delete[] questions;
    questions = new Question[numQuestions];
}

//
// C++ Interface: question
//
// Description: Заголовок класса для вопросов
//
//
// Author: Ad_Astra <ad-astra@ad-astra.name>, (C) 2008
//
// Copyright: See COPYING file that comes with this distribution
//
//
#ifdef QUESTION_H
#define QUESTION_H
#include <QCoreApplication>
#include "answer.h"

class Question {
public:
    Question();
    Question(const int numAns);
    Question(const int numAns, const QString *q);
    ~Question();

    void setText(const QString *q);
    void addAnswer(const Answer *a);

    QString getText();
    Answer* getAnswerByNum(int num);
    int countAnswers();

private:
    QString text;
    int numAnswers;
    Answer *answers;
    void shuffle();
    int currIndex;
};
#endif

//
// C++ Implementation: question
//
// Description: Класс для вопросов. Каждый вопрос содержит несколько ответов
// (их количество может быть произвольным, т.к. массив ответов создаётся
// динамически) и текст вопроса.
//
//
// Author: Ad_Astra <ad-astra@ad-astra.name>, (C) 2008
//

```

```

// Copyright: See COPYING file that comes with this distribution
//
//
#include <QCoreApplication>
#include <cstdlib>
#include "answer.h"
#include "question.h"

using namespace std;
/**
 * Это базовый конструктор класса, создающий вопрос с пустым текстом и пустым
 * списком ответов
 */
Question::Question() : text(""), numAnswers(0), currIndex(0), answers(0) {}
/**
 * Конструктор класса, создающий вопрос с пустым текстом и заданным через
 * параметр количеством ответов
 * @param numAns количество ответов
 */
Question::Question(const int numAns) : text(""), numAnswers(numAns),
currIndex(0), answers(new Answer[numAns]) {}
/**
 * Наиболее параметризованный конструктор класса, создающий вопрос с заданным
 * текстом
 * и заданным количеством ответов
 * @param numAns количество ответов
 * @param q текст вопроса - объект типа QString
 */
Question::Question(const int numAns, const QString *q) : text(*q),
numAnswers(numAns), currIndex(0), answers(new Answer[numAns]) {}
/**
 * Это деструктор класса, призванный при уничтожении объекта с вопросом удалить
 * из памяти также все принадлежащие ему ответы
 */
Question::~Question() {
    delete[] answers;
}

/**
 * Эта функция задаёт текст вопроса
 * @param q указатель на строку типа QString
 */
void Question::setText(const QString *q) {
    text = *q;
}

/**
 * Эта функция выводит текст вопроса
 * @return текст вопроса
 */
QString Question::getText() {
    return text;
}

/**
 * Эта функция считает ответы, принадлежащие данному вопросу
 * @return число ответов
 */
int Question::countAnswers() {
    return numAnswers;
}

```

```

}

/**
 * Эта функция добавляет вариант ответа к имеющемуся вопросу, после чего
 * вызывает перетасовку ответов, дабы порядок не оставался неизменным
 * @param a - указатель на объект типа Answer
 */
void Question::addAnswer(const Answer *a) {
    answers[currIndex] = *a;
    currIndex++;
    shuffle();
}

/**
 * Эта функция определяет ответ, идущий под заданным номером в списке ответов,
 * принадлежащих данному вопросу
 * @param num - номер вопроса
 * @return указатель на объект типа Answer
 */
Answer* Question::getAnswerByNum(int num) {
    return &answers[num];
}

/**
 * Эта функция перетасовывает ответы, чтобы те шли в случайном порядке
 */
void Question::shuffle() {
    for (int cnt=0; cnt < currIndex; cnt++) {
        int randNum = cnt + (rand() % (currIndex-cnt));
        Answer tmpAnswer = answers[cnt];
        answers[cnt] = answers[randNum];
        answers[randNum] = tmpAnswer;
    }
}

}

//
// C++ Interface: questWidget
//
// Description: Заголовок класса виджета для вариантов ответов
//
//
// Author: Ad_Astra <ad-astra@ad-astra.name>, (C) 2009
//
// Copyright: See COPYING file that comes with this distribution
//
//
#ifndef QUESTWIDGET_H
#define QUESTWIDGET_H

#include <QtGui>
#include <QCoreApplication>
#include "question.h"
class QuestWidget : public QWidget {
    Q_OBJECT

public:
    QuestWidget(Question *q, QWidget *parent = 0);
    ~QuestWidget();
    int getChoice();
private slots:

```

```

        void setChoice(int selChoice);
private:
        int choice;
        QButtonGroup *radios;
};
#endif

//
// C++ Implementation: questwidget
//
// Description: Эта штукавина генерирует однотипные виджеты для вопросов и
// вариантов ответов, дабы не засорять главный класс формы, а также позволяет
// удобно и без лишнего ковыряния в "кишках" получать информацию о выбранном
// элементе
//
//
// Author: Ad_Astra <ad-astra@ad-astra.name>, (C) 2009
//
// Copyright: See COPYING file that comes with this distribution
//
//
#include <QtGui>
#include <QCoreApplication>
#include "questwidget.h"
#include "question.h"
#include "answer.h"

/**
 * Конструктор класса, собирающий группу радиокнопок с вариантами ответов и
 * рамкой с текстом вопроса "на лету" из вопроса в параметре
 * @param q объект класса Question
 * @param parent
 */
QuestWidget::QuestWidget(Question *q, QWidget *parent)
    : QWidget(parent), choice(-1), radios (new QButtonGroup)
{
    connect(radios, SIGNAL(buttonClicked(int)), this, SLOT(setChoice(int)));

    QGroupBox *groupBox = new QGroupBox(q->getText());
    QVBoxLayout *group = new QVBoxLayout;
    for(int cnt = 0; cnt<=q->countAnswers()-1; cnt++) {
        Answer *ans = q->getAnswerByNum(cnt);
        QRadioButton *option = new QRadioButton(ans->getText());
        radios->addButton(option, cnt);
        group->addWidget(option);
    }
    groupBox->setLayout(group);
    QVBoxLayout *layout = new QVBoxLayout;
    layout->addWidget(groupBox);
    setLayout(layout);
}

/**
 * Деструктор класса, нужный для того, чтобы при удалении виджета удалялся из
 * памяти и набор принадлежащих ему радио-кнопок с вариантами ответов
 */
QuestWidget::~QuestWidget() {
    delete radios;
}

```

```

/**
 * Эта функция вызывается при щелчке по радио-кнопке в группе вопросов,
 * запоминая свежевывбранный элемент
 * @param selChoice индекс выбранной радио-кнопки
 */
void QuestWidget::setChoice(int selChoice) {
    choice = selChoice;
}

/**
 * Эта функция позволяет определить, какой из вариантов ответа выбран
 * @return номер выбранного ответа
 */
int QuestWidget::getChoice() {
    return choice;
}

//
// C++ Interface: answer
//
// Description: заголовок класса для ответов
//
//
// Author: Ad_Astra <ad-astra@ad-astra.name>, (C) 2008
//
// Copyright: See COPYING file that comes with this distribution
//
//
#ifdef ANSWER_H
#define ANSWER_H
#include <QCoreApplication>

class Answer {
public:
    Answer();
    Answer(const QString *a);
    Answer(const QString *a, bool c);
    bool isCorrect();
    QString getText();

private:
    bool correct;
    QString text;
};
#endif

//
// C++ Implementation: answer
//
// Description: Это класс для ответов. Он очень простой, и содержит лишь текст
// ответа и логическую переменную, определяющую, верен ли он.
//
//
// Author: Ad_Astra <ad-astra@ad-astra.name>, (C) 2008
//
// Copyright: See COPYING file that comes with this distribution
//
//
#include <QCoreApplication>
#include "answer.h"

```

```

/**
 * Это базовый конструктор, создающий неправильный ответ с пустым текстом
 */
Answer::Answer() : correct(false), text("") {}
/**
 * Это основной конструктор, создающий неправильный ответ с заданным текстом
 * @param a - текст вопроса
 */
Answer::Answer(const QString *a) : correct(false), text(*a) {}
/**
 * Это наиболее настраиваемый конструктор, позволяющий указать как текст
 * вопроса,
 * так и правильность ответа
 * @param a - текст вопроса
 * @param c - правильность ответа
 */
Answer::Answer(const QString *a, bool c) : correct(c), text(*a) {}

/**
 * Эта функция, будучи вызванной для конкретного ответа, определяет, верен ли он
 * @return true, если ответ верен, false в противном случае
 */
bool Answer::isCorrect() {
    return correct;
}

/**
 * Эта функция выводит текст ответа
 * @return текст ответа в переменной типа QString
 */
QString Answer::getText() {
    return text;
}

//
// C++ Interface: highscore
//
// Description: Заголовок класса формы результатов
//
//
// Author: Ad_Astra <ad-astra@ad-astra.name>, (C) 2009
//
// Copyright: See COPYING file that comes with this distribution
//
//
#ifdef HIGHSCORE_H
#define HIGHSCORE_H
#include "ui_high_score.h"

class HighScoreWindow : public QDialog
{
    Q_OBJECT

public:
    HighScoreWindow(QWidget *parent = 0, QString* fileName=new QString(' '));
    ~HighScoreWindow();

private slots:

private:

```

```

        Ui_HighScoreWindow ui;
        QTableWidgetItem *newUName;
        QTableWidgetItem *newAtts;
        QTableWidgetItem *newScore;
};

#endif

//
// C++ Implementation: highscore
//
// Description: Это класс формы результатов. Форма результатов - простенькое
// модальное диалоговое окно, единственным своим виджетом содержащее таблицу
// для вывода результатов.
//
//
// Author: Ad_Astra <ad-astra@ad-astra.name>, (C) 2009
//
// Copyright: See COPYING file that comes with this distribution
//
//
#include "mainform.h"
#include "highscore.h"
#include <QtGui>
#include <QCoreApplication>
#include <QtXml>
/**
 * Это конструктор формы результатов. Собственно, его достаточно - главное,
 * заметить, что к его вызову добавлен дополнительный параметр - имя файла с
 * результатами тестов, из которого будут получены данные и, соответственно,
 * выведены в таблицу.
 * @param parent
 * @param fileName
 */
HighScoreWindow::HighScoreWindow(QWidget *parent, QString* fileName)
    : QDialog(parent), newScore(0), newAtts(0), newUName(0)
{
    ui.setupUi(this);

    QFile *resultsFile = new QFile("./results/" + *fileName + ".xml");
    if (!resultsFile->open(QIODevice::ReadOnly)) {
        QMessageBox::warning(0, QString::fromUtf8("Ошибка!"),
        QString::fromUtf8("Этот тест ещё никто не проходил"));
    } else {
        QDomDocument* resultsXML = new QDomDocument();
        resultsXML->setContent(resultsFile);

        QDomNodeList userList = resultsXML->elementsByTagName("user");
        if (userList.count() > 0) {
            for (int cnt = 0; cnt < userList.count(); cnt++) {
                ui.hsTable->setRowCount(ui.hsTable->rowCount() + 1);
                newUName = new
        QTableWidgetItem(userList.item(cnt).toElement().attribute("name"));

                QDomNodeList attempts = userList.item(cnt).childNodes();
                int attemptsNum = attempts.count();
                newAtts = new QTableWidgetItem(QString::number(attemptsNum));
                int topScore=0;
                for (int attCount=0; attCount < attemptsNum; attCount++) {
                    if
        (attempts.item(attCount).toElement().attribute("score").toInt() > topScore) {

```

```

        topScore =
attempts.item(attCount).toElement().attribute("score").toInt();
    }
    }
    newScore = new QTableWidgetItem(QString::number(topScore));

    ui.hsTable->setItem(cnt, 2, newScore);
    ui.hsTable->setItem(cnt, 1, newAtts);
    ui.hsTable->setItem(cnt, 0, newUName);
    }
}
resultsFile->close();
delete resultsXML;
}
delete resultsFile;
}

/**
 * Это деструктор класса. Он освобождает вспомогательные переменные,
 * использующиеся для построения таблицы.
 */
HighScoreWindow::~HighScoreWindow() {
    delete newScore;
    delete newAtts;
    delete newUName;
}

```