|  |
| --- |
|  |
| МИНОБРНАУКИ РОССИИ |
| *Федеральное государственное бюджетное образовательное учреждение высшего образования* ***«МИРЭА – Российский технологический университет»***  **РТУ МИРЭА** |

**Институт информационных технологий (ИИТ)**

**Кафедра инструментального и прикладного программного обеспечения (ИиППО)**

**КУРСОВАЯ РАБОТА**

по дисциплине: Объектно-ориентированное программирование

по профилю: Система поддержки принятия решений

направления профессиональной подготовки: (09.03.04) Программная инженерия, бакалавриат

Тема: «Разработка программы для ведения списка дел»

Студент:

Группа:

Работа представлена к защите\_\_\_\_\_\_\_\_\_\_\_(дата)\_\_\_\_\_\_\_\_\_/ /

Руководитель:

Работа допущена к защите\_\_\_\_\_\_\_\_\_\_\_(дата)\_\_\_\_\_\_\_\_\_/ /

Оценка по итогам защиты: \_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ / \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ /

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ / \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ /

(подписи, дата, ф.и.о., должность, звание, уч. степень двух преподавателей, принявших защиту)

М. МИРЭА. 2019г.

|  |
| --- |
|  |
| МИНОБРНАУКИ РОССИИ |
| *Федеральное государственное бюджетное образовательное учреждение высшего образования* ***«МИРЭА – Российский технологический университет»***  **РТУ МИРЭА** |

**Институт информационных технологий (ИТ)**

**Кафедра инструментального и прикладного программного обеспечения (ИиППО)**

**ЗАДАНИЕ НА ВЫПОЛНЕНИЕ КУРСОВУЮ РАБОТЫ**

по дисциплине: Объектно-ориентированное программирование

по профилю: Система поддержки принятия решений

направления профессиональной подготовки: 09.03.04 Программная инженерия, бакалавриат

Студент:

Группа:

Срок представления к защите \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Руководитель:

Тема: «Разработка программы для ведения списка дел»

Исходные данные: IDE Qt Creator 4.6.1 (Community)

Перечень вопросов к разработке (основная задача), графические материалы (слайды):

1. Определение функциональных назначений разрабатываемой системы, определение состава прототипа
2. Проектирование UML-диаграммы классов системы
3. Проектирование алгоритмических решений для прототипа
4. Реализация разработанных решений
5. Оформление пояснительной записки
6. Разработка презентации

Руководителем произведён инструктаж по технике безопасности, противопожарной технике и правилам внутреннего распорядка.

Заведующий кафедрой ИиППО: \_\_\_\_\_\_\_\_\_\_ /В.А. Мордвинов/,

дата: \_\_\_\_\_\_\_\_

Задание на КР выдал \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ ()

«\_\_\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_2019г.

Задание на КР получил\_\_\_\_\_\_\_\_\_\_\_\_\_ ()

«\_\_\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_2019г.

Оглавление

[ВВЕДЕНИЕ 5](#_Toc11343511)

[1 ТЕОРЕТИЧЕСКАЯ ЧАСТЬ 8](#_Toc11343512)

[1.1 Язык программирования C++ 8](#_Toc11343513)

[1.2 Библиотека шаблонов Qt 8](#_Toc11343514)

[1.2.1 Общие сведения 8](#_Toc11343515)

[1.2.2 Описание используемых контейнеров 9](#_Toc11343516)

[1.3 Краткое описание используемой среды разработки 9](#_Toc11343517)

[2 ПРАКТИЧЕСКАЯ ЧАСТЬ 10](#_Toc11343518)

[2.1 Постановка задачи 10](#_Toc11343519)

[2.2 Проектное решение 10](#_Toc11343520)

[2.3 Реализация программы 12](#_Toc11343521)

[ЗАКЛЮЧЕНИЕ 19](#_Toc11343522)

[СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ 20](#_Toc11343523)

[ПРИЛОЖЕНИЕ А. ЛИСТИНГ ПРОГРАММЫ 21](#_Toc11343524)

# ВВЕДЕНИЕ

В настоящее время, с развитием научно-технического прогресса и информационных технологий, сложность автоматизируемых предметных областей постоянно и неуклонно возрастает. Программист проводит всё своё рабочее время, по сути, в борьбе со сложностью программ, отлаживая свой код в поисках допущенных ошибок. Ради борьбы со сложностью эволюционируют и языки программирования.

После машинных кодов и перфокарт появились текстовые редакторы и язык ассемблера, что позволило писать чуть более сложные и объёмные программы при меньших трудозатратах. Далее появились процедурные языки высокого уровня – C, Паскаль и многие другие, что перевело технологии программирования на новый уровень.

Следующим крупным скачком на пути борьбы со сложностью стало появление объектно-ориентированного подхода и разработка соответствующих языков и компиляторов. Так C эволюционировал в C++. Однако прогресс и сопутствующий ему рост сложности информационных систем не останавливаются на месте и вот появляются всё новые подходы к проектированию программ. Акцент при разработке сложных программных систем смещается с собственно написания кода в сторону проектирования и моделирования, что неизбежно приводит к совершенствованию соответствующих методик. При проектировании всегда возникает вопрос, как перенести на бумагу и электронной носитель идеи, которые возникают в голове разработчика – а главное, чтобы донести эти идеи в понятной форме до коллег.

К концу прошлого века сформировалась критическая масса различных подходов к анализу и разработке сложных информационных систем. Эволюция различных подходов привела к осознанию необходимости объединения их в один, чтобы совместными усилиями развивать и совершенствовать универсальный подход с моделированию предметной области. В результате появился универсальный язык моделирования, UML (англ. Unified Modeling Language). В 1990 году была представлена его официальная версия 1.0. В настоящее время эволюция UML также не стоит на месте - в августе 2005 года опубликована формальная спецификация 2.0, а в июне 2015 года – версия 2.5.

Данная работа посвящена изучению современных подходов к разработке сложных информационных систем, начиная от их моделирования и заканчивая разработкой готового продукта на языке высокого уровня. В качестве языка программирования выбран С++ с использованием в качестве среды разработки бесплатно распространяемой системы QtCreator (Community).

Целью работы является закрепление и углубление знаний, полученных при изучении языка C++, а также развитие практических навыков при выборе представления исходных данных, использовании стандартных библиотек контейнеров при написании программ, тестировании, отладки программы и оформлении сопроводительной документации.

В качестве предметной области для демонстрации возможностей C++ в тандеме с платформой Qt выбрана записная книжка со списком дел, в которой пользователь может просматривать свои задачи по дням, а также добавлять новые, редактировать и удалять существующие дела и задачи.

# ТЕОРЕТИЧЕСКАЯ ЧАСТЬ

## Язык программирования C++

Предшественником языка C++ является императивный (процедурный) язык программирования C. Язык C++ значительно расширяет возможности своего предшественника и включает в себя сразу три парадигмы – процедурное, объектно-ориентированное и обобщенное программирование. При этом соблюдается обратная совместимость с предшественником. По сути, современный язык C++ – это три диалекта, весьма существенно отличающиеся друг от друга.

В последние годы, с внедрением стандартов 2011, 2014 и 2017 годов, наметилась тенденция к появлению четвёртого диалекта. Речь идёт о так называемом функциональном программировании, в котором, в противовес императивной парадигме, процесс вычисления трактуется как последовательное вычисление значений функций в математическом понимании, а не смена состояний в результате череды процедурных вызовов.

## Библиотека шаблонов Qt

### Общие сведения

Как известно, стандартная библиотека шаблонов (STL – Standard template library), поставляемая в комплекте с компилятором, включает наиболее распространённые контейнеры и алгоритмы.

Механизм шаблонов встроен в компилятор языка C++, чтобы дать возможность программистам делать свой код короче за счет обобщенного программирования – речь идёт об упоминавшейся ранее парадигме обобщенного программирования. Именно на этой парадигме и строится стандартная библиотека шаблонов STL. Самые популярные коллекции из библиотеки это string, vector, list, queue, stack, map (multimap) и set (multiset).

В платформу Qt встроена библиотека шаблонов, аналогичная STL. Сделано это разработчиками Qt для того, чтобы не зависеть от конкретной реализации STL разными поставщиками компиляторов, которая далеко не всегда бывает удовлетворительной. Узнаваемые по STL имена классов предваряются заглавной буквой «Q».

Основные контейнерные классы – QString, QVector, QList, QStack, QQueue, ассоциативные контейнеры (словари) – QSet, QMap, QMultiMap, QHash, QMultiHash. Все классы являются шаблонами, параметризируемыми хранимым типом данных и типом данных ключа, для словарей.

### Описание используемых контейнеров

В работе использованы такие контейнеры как QString, QVector и QList. Контейнер QString используется в программе для хранения всех строк. Этот шаблонный класс, точно так же как и std::string библиотеки STL, является обобщением только для строковых последовательностей – как правило, это строки из символов типа char и wchar\_t («Wide Char»). Он инкапсулирует в себе все рутинные операции по управлению памятью и является крайне удобной заменой использовавшимся в язык C символьным массивам. Контейнеры QVector и QList так же являются аналогами одноимённых шаблонов из состава STL. QVector используется для хранения общего списка дел, а QList – для хранения названий подарков на день рождения.

## Краткое описание используемой среды разработки

Интегрированная среда разработки Qt Creator (ранее известная под названием Greenhouse) – [кроссплатформенная](https://ru.wikipedia.org/wiki/%D0%9A%D1%80%D0%BE%D1%81%D1%81%D0%BF%D0%BB%D0%B0%D1%82%D1%84%D0%BE%D1%80%D0%BC%D0%B5%D0%BD%D0%BD%D0%BE%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%BD%D0%BE%D0%B5_%D0%BE%D0%B1%D0%B5%D1%81%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%B8%D0%B5), [свободно](https://ru.wikipedia.org/wiki/%D0%A1%D0%B2%D0%BE%D0%B1%D0%BE%D0%B4%D0%BD%D0%BE%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%BD%D0%BE%D0%B5_%D0%BE%D0%B1%D0%B5%D1%81%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%B8%D0%B5) распространяемая [IDE](https://ru.wikipedia.org/wiki/%D0%A1%D1%80%D0%B5%D0%B4%D0%B0_%D1%80%D0%B0%D0%B7%D1%80%D0%B0%D0%B1%D0%BE%D1%82%D0%BA%D0%B8_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%BD%D0%BE%D0%B3%D0%BE_%D0%BE%D0%B1%D0%B5%D1%81%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%B8%D1%8F) (Integrated Development Environment) для разработки на С, С++ и [QML](https://ru.wikipedia.org/wiki/QML). Разработана компанией Trolltech (Digia) для работы с платформой [Qt](https://ru.wikipedia.org/wiki/Qt). Включает в себя графический интерфейс для разработки и отладки исходного кода программ как с использованием QtWidgets, так и QML. Поддерживаемые компиляторы: [GCC](https://ru.wikipedia.org/wiki/GCC), [Clang](https://ru.wikipedia.org/wiki/Clang), [MinGW](https://ru.wikipedia.org/wiki/MinGW), [MSVC](https://ru.wikipedia.org/wiki/MSVC), [Linux ICC](https://ru.wikipedia.org/wiki/Intel_C%2B%2B_compiler), [GCCE](https://ru.wikipedia.org/w/index.php?title=GCCE&action=edit&redlink=1), [RVCT](https://ru.wikipedia.org/w/index.php?title=RVCT&action=edit&redlink=1), [WINSCW](https://ru.wikipedia.org/w/index.php?title=WINSCW&action=edit&redlink=1)[[3]](https://ru.wikipedia.org/wiki/Qt_Creator#cite_note-3).

# ПРАКТИЧЕСКАЯ ЧАСТЬ

## Постановка задачи

В ходе работы необходимо разработать программу, работающую под управлением операционной системы Windows, имеющую графический пользовательский интерфейс и позволяющую вести список запланированных дел – добавлять, редактировать и удалять задачи, встречи, звонки и дни рождения. Программа предназначена для персонального использования.

Язык программирования – C++, среда разработки – Qt Creator, позволяющая выбирать конкретный компилятор (под Windows это, как правило, MinGW (семейство компиляторов gcc) или Microsoft Visual C++).

Наименование программы – «Личный календарь».

## Проектное решение

Основной сущностью этой программной системы является «Календарь», которая, очевидно, будет контейнером для вспомогательных сущностей, каждая из которых будет представлять собой один из нескольких типов запланированных дел. «Дела» могут быть разными – это встреча с друзьями, коллегами, партнёрами по бизнесу и т.д., звонки кому-либо, дни рождения, с которых не хочется забыть, и многое другое.

Базовыми характеристиками сущности «Дело» являются дата/время, на которые оно запланировано, а также название. Дата/время представлено в исходном коде программы полем типа QDateTime, а название – строковым типом QString. Одной этой сущности достаточно для планирования широкого спектра дел, не требующих дополнительной информации. Для реализации этой сущности в программе определён базовый класс Task.

Но для ряда запланированных задач удобно придать этой сущности дополнительные характеристики. Например, если человек собирается позвонить, то необходимо указать, кому именно – причём удобно хранить эту информацию не в названии, а в отдельном поле, чтобы, например, в ходе дальнейшего развития программного продукта можно было внедрить сортировку и поиск для конкретных видов дел (задач). То же самое касается и других перечисленных выше дел – для встречи, кроме визави, желательно указать ещё и место, где она состоится, а для дня рождения удобно бывает сохранить запланированный заранее список подарков.

Для всех этих категорий выделены отдельные сущности, которые последовательно детализируют своих предшественников, добавляя всё новые характеристики.

Ближайшим наследником «Дела» является сущность «Взаимодействие», которая содержит такую характеристику как «С кем» и подходит для планирования широкого круга задач, в которых, кроме самого пользователя программы, задействован другой человек. В демонстрационной программе, разрабатываемой в данной работе, основное внимание уделено запланированным звонкам, что отражено в текстовом сопровождении этой сущности в пользовательском интерфейсе. Поскольку в данной реализации мы не управляем отдельным списком контактов, для этого поля выбран строковый тип QString. Для реализации самой этой сущности в программе определён класс Interaction, наследующий от класса Task.

«Взаимодействие» уточняется двумя другими сущностями – «Встреча» и «День рождения», которые к понятию «С кем» добавляют дополнительную информацию. «Встреча» содержит строковую (QString) характеристику «Где», определяющую место встречи, а «День рождения» содержит такую характеристику как «Список подарков». Для представления списка подарков выбран контейнер QList с последовательным доступом, поскольку не предвидится необходимость произвольного доступа к каждому конкретному подарку в этом списке. Каждый подарок, в свою очередь, представлен строковым (QString) элементом этого контейнера. Для реализации сущностей «Встреча» и «День рождения» в программе определены классы, являющиеся прямыми потомками класса Interaction – «Встреча» представлена классом Meeting, а «День рождения» – классом Birthday.

UML-диаграмма вышеописанных классов приведена на рисунке 1.

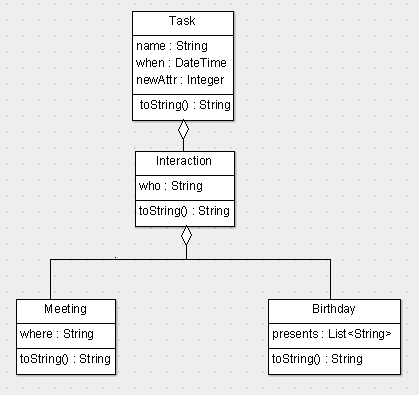


Рисунок 1 – UML-диаграмма классов

Визуальная часть программы представлена типовыми классами, наследующими от классов графической библиотеки Qt. В данном приложении это MainWindow, отвечающий за работу главного окна и TaskDlg, управляющий диалоговым окном с параметрами планируемой задачи.

## Реализация программы

Скомпилированная программа представляет собой исполняемый файл calendar.exe и запускается на выполнение из командной строки, из проводника Windows – непосредственно через этот файл или через пиктограмму ярлыка, который на него ссылается, или непосредственно из интегрированной среды разработки QtCreator.

После запуска программы откроется главное окно программы, изображённое на рисунке 2.

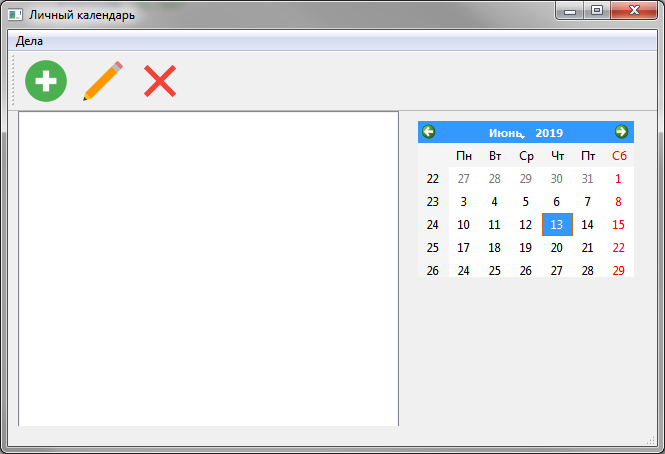


Рисунок 2 – Главное окно программы

Доступные пользователю действия представлены в виде пунктов меню, показанных на рисунке 2, а также пиктограммами на панели инструментов, расположенной в верхней части программы, сразу под строкой меню.

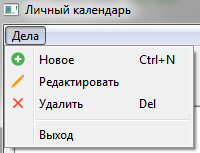


Рисунок 3 – Вид главного меню

В правой части окна программы расположен элемент управления «Календарь», который позволяет выбрать текущую дату. Основную часть окна занимает список задач, запланированных на этот день – на рисунке 2 он пуст, поскольку не добавлено ещё ни одного дела.

При добавлении новой задачи откроется диалоговое окно, где будет предложено выбрать тип и параметры добавляемого события. Вид диалога при добавлении нового дела показан на рисунке 4.

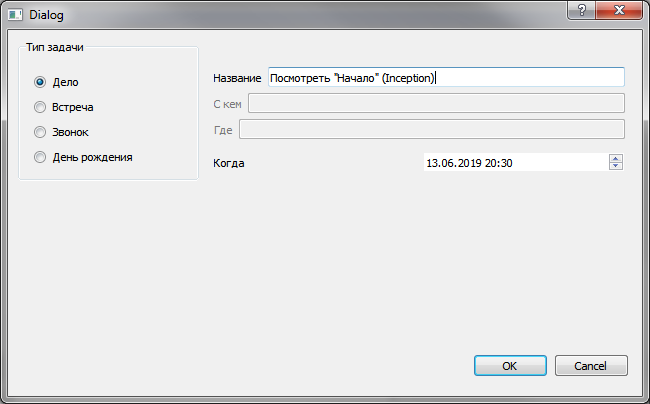


Рисунок 4 – Диалог добавления общей задачи

Как видно из рисунка, в левой части диалога размещена область элементов выбора типа задачи – «Дело», «Встреча», «Звонок» и «День рождения». Поля «Название» и «Когда» доступны пользователю всегда, поскольку отвечают за два базовых атрибута задания. Поле «Когда» сразу после открытия диалогового окна инициализируется текущими датой и временем, которые впоследствии можно менять в соответствии с планами. Остальные поля ввода отображаются в зависимости от выбранного типа задачи.

Вид диалога при добавлении встречи показан на рисунке 5.

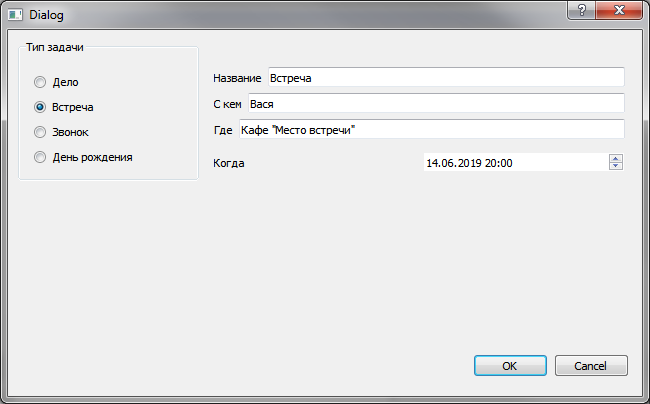


Рисунок 5 – Диалог добавления встречи

Для встречи доступными становятся поля «С кем» и «Где».

Вид диалога при добавлении планируемого звонка показан на рисунке 6.

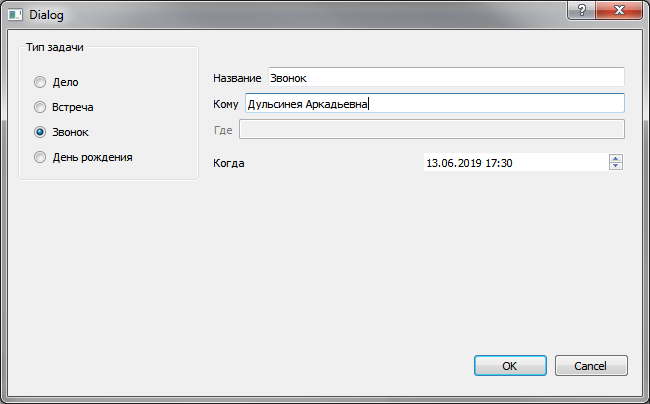


Рисунок 6 – Диалог добавления звонка

В этом случае доступно только дополнительное поле ввода «Кому».

Вид диалога при добавлении дня рождения показан на рисунке 7.

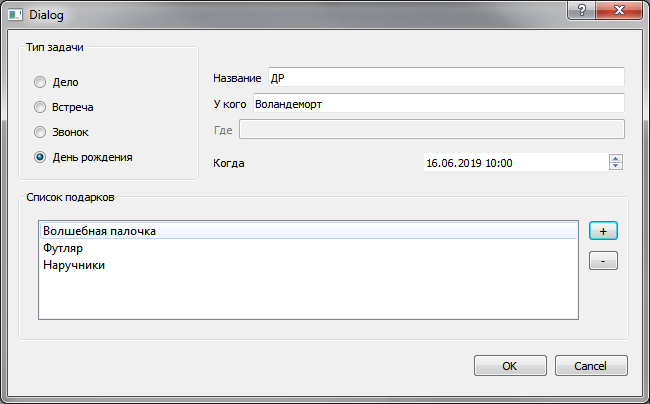


Рисунок 7 – Диалог добавления дня рождения

В этом режиме добавления можно указать, чей это день рождения и какие подарки планируется подарить. Подарки отображается в списке внизу экрана. Справа от списка расположены кнопки для добавления и удаления подарков, при добавлении отображается простой диалог, в котором можно внести наименование подарка, как показано на рисунке 8.

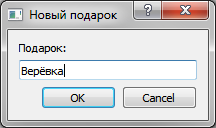


Рисунок 7 – Диалог добавления подарка на день рождения

В нижней части всех диалогов расположены стандартные кнопки «OK» и «Cancel», которые закрывают диалог, причём первая применяет пользовательское действие и отображает изменения в редактируемом списке, а вторая – отменяет.

После добавления того или иного дела, оно отобразится в списке на главной форме приложения, с учётом выбранного фильтра по текущей дате, как показано на рисунке 9, на примере 13 июня 2019 г.

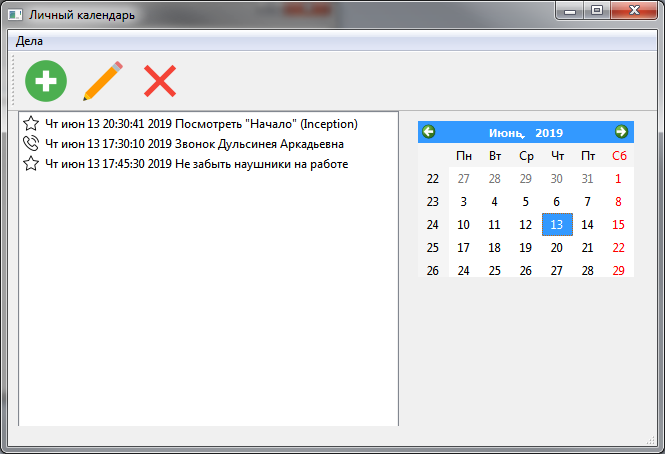


Рисунок 9 – Главное окно программы со списком дел на 13.06.2019

При выборе другой любой другой даты, отобразится другой список (возможно, пустой) – список тех дел, которые запланированы на тот день. Отличия отображены на рисунке 10, где выбрана дата 16 июня 2019 г.

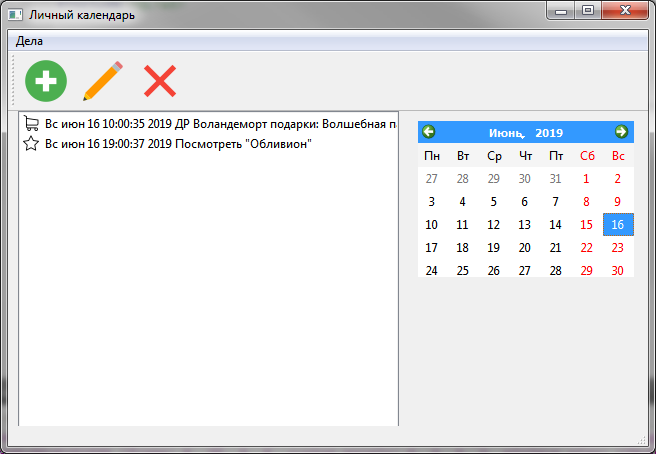


Рисунок 10 – Главное окно программы со списком дел на 16.06.2019

Все представленные в программе списочные элементы управления – дел и подарков – наряду с добавлением поддерживают также операцию удаления. В зависимости от типа конкретного дела, его элемент в списке сопровождается соответствующей пиктограммой. Для того чтобы удалить элемент из списка, нужно выделить его и нажать соответствующую кнопку или команду на панели инструментов или в меню.

Кроме того, список дел поддерживает также операцию редактирования. В этом случае открывается то же самое диалоговое окно, что и при добавлении, только все поля уже автоматически заполнены соответствующими параметрами редактируемой задачи.

# ЗАКЛЮЧЕНИЕ

В результате работы создана программа, позволяющая вести календарный список запланированных дел – задач, встреч, звонков и поздравлений. Эта программа демонстрирует применение в повседневной разработке языка C++, а также контейнеров библиотеки Qt – аналогов стандартной библиотеки языка. В результате работы также продемонстрированы возможности мощной архитектурной платформы Qt и её редактора кода – Qt Creator.

Приобретены практические навыки разработки приложений, использования языковых конструкций С++ и инструментов Qt, отладки программ и составления сопроводительной документации. На базе практического применения, закреплены базовые знания, полученные ранее, при изучении теории.

# СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ

1. Николаи М. Джосаттис Стандартная библиотека C++. Справочное руководство / Николаи М. Джосаттис. - М.: Вильямс, 2014. – 123 c.

2. Герб Саттер Решение сложных задач на С++ / Герб Саттер. - М.: Вильямс, 2015. – 400 c.

3. Роберт Лафоре Объектно-ориентированное программирование в С++ / Роберт Лафоре. - М.: Питер, 2015. – 928 c.

4. Васильев А.Н. Самоучитель С++ с примерами и задачами. 4-е издание (переработанное). — СПб.: Наука и Техника, 2016. – 480 с.

5. Вставская Е. UML-диаграммы классов / PROG-CPP.RU: Программирование на Си и С++ – 17.10.2017 – URL: http://prog-cpp.ru/uml-classes/ (дата обращения: 06.05.2018).

6. ГОСТ 2.105-95 Общие требования к текстовым документам.

# ПРИЛОЖЕНИЕ А. ЛИСТИНГ ПРОГРАММЫ

task.h

#ifndef TASK\_H

#define TASK\_H

#include <QString>

#include <QDateTime>

enum TaskType {

ttCommon,

ttMeeting,

ttCall,

ttBirthday

};

class Task

{

public:

Task();

Task(const QDateTime& when, const QString& name);

// Виртуальный деструктор нужен для того, чтобы

// компилятор сформировал таблицу виртуальных функций

// и при удалении экземпляра одного из потомков

// корректно вызывал деструкторы для всеё цепочки наследования

virtual ~***Task***() = default;

virtual QString ***toString***() const;

QString **name**() const {

return m\_name;

}

void **setName**(const QString& name) {

m\_name = name;

}

QDateTime **when**() const {

return m\_when;

}

void **setWhen**(const QDateTime& when) {

m\_when = when;

}

private:

QString m\_name;

QDateTime m\_when;

};

#endif // TASK\_H

task.cpp

#include "task.h"

Task::**Task**()

{

}

Task::**Task**(const QDateTime& when, const QString& name)

: m\_name(name), m\_when(when)

{

}

QString Task::***toString***() const

{

return m\_when.toString() + " " + m\_name;

}

interaction.h

#ifndef INTERACTION\_H

#define INTERACTION\_H

#include "task.h"

class Interaction : public Task

{

public:

Interaction();

Interaction(const QDateTime& when, const QString& name, const QString& who);

QString ***toString***() const override;

QString **who**() const {

return m\_who;

}

void **setWho**(const QString& who) {

m\_who = who;

}

private:

QString m\_who;

};

#endif // INTERACTION\_H

interaction.cpp

#include "interaction.h"

Interaction::**Interaction**(): Task()

{

}

Interaction::**Interaction**(const QDateTime& when, const QString& name, const QString& who)

: Task(when, name), m\_who(who)

{

}

QString Interaction::***toString***() const

{

return Task::*toString*() + " " + m\_who;

}

meeting.h

#ifndef MEETING\_H

#define MEETING\_H

#include "interaction.h"

class Meeting : public Interaction

{

public:

Meeting();

Meeting(const QDateTime& when, const QString& name, const QString& who, const QString& where);

QString ***toString***() const override;

QString **where**() const {

return m\_where;

}

void **setWhere**(const QString& where) {

m\_where = where;

}

private:

QString m\_where;

};

#endif // MEETING\_H

meeting.cpp

#include "meeting.h"

Meeting::**Meeting**(): Interaction()

{

}

Meeting::**Meeting**(const QDateTime& when, const QString& name, const QString& who, const QString& where)

: Interaction(when, name, who), m\_where(where)

{

}

QString Meeting::***toString***() const

{

return Interaction::*toString*() + " место: " + where();

}

birthday.h

#ifndef BIRTHDAY\_H

#define BIRTHDAY\_H

#include "interaction.h"

#include <QList>

class Birthday : public Interaction

{

public:

Birthday();

Birthday(const QDateTime& when, const QString& name, const QString& who);

QString ***toString***() const override;

QList<QString> **presents**() const {

return m\_presents;

}

void **addPresent**(const QString& present) {

m\_presents.push\_back(present);

}

private:

QList<QString> m\_presents;

};

#endif // BIRTHDAY\_H

birthday.cpp

#include "birthday.h"

Birthday::**Birthday**(): Interaction()

{

}

Birthday::**Birthday**(const QDateTime& when, const QString& name, const QString& who)

: Interaction(when, name, who)

{

}

QString Birthday::***toString***() const

{

QString result = Interaction::*toString*();

result += " подарки:";

for(auto gift : m\_presents) {

result += (" " + gift);

}

return result;

}

calendar.h

#ifndef CALENDAR\_H

#define CALENDAR\_H

#include <QMap>

#include <QVector>

class Task;

class Calendar

{

public:

Calendar() = default;

~Calendar();

Task\* **task**(int index) const;

bool **setTask**(int index, Task\* task);

bool **delTask**(int index);

void **addTask**(Task\* task);

int **taskCount**() const {

return m\_tasks.size();

}

private:

QVector<Task\*> m\_tasks;

};

#endif // CALENDAR\_H

calendar.cpp

#include "calendar.h"

#include "task.h"

#include <QDebug>

Calendar::~Calendar()

{

for(auto task : m\_tasks) {

if(task) {

delete task;

}

}

}

Task\* Calendar::**task**(int index) const

{

if(index < 0 || index >= taskCount()) {

return 0;

}

return m\_tasks[index];

}

bool Calendar::**setTask**(int index, Task\* task)

{

if(index < 0 || index >= taskCount()) {

return false;

}

if(m\_tasks[index]) {

delete m\_tasks[index];

}

m\_tasks[index] = task;

return true;

}

bool Calendar::**delTask**(int index)

{

if(index < 0 || index >= taskCount()) {

return false;

}

if(m\_tasks[index]) {

delete m\_tasks[index];

}

qDebug() << "deleting " + QString::number(index) + " count " + QString::number(m\_tasks.size());

m\_tasks.removeAt(index);

qDebug() << "done, count " + QString::number(m\_tasks.size());

return true;

}

void Calendar::**addTask**(Task\* task)

{

m\_tasks.push\_back(task);

}

mainwindow.h

#ifndef MAINWINDOW\_H

#define MAINWINDOW\_H

#include <QMainWindow>

#include <QDate>

namespace Ui {

class MainWindow;

}

class Calendar;

class Task;

class MainWindow : public QMainWindow

{

Q\_OBJECT

public:

explicit MainWindow(QWidget \*parent = 0);

~***MainWindow***();

private Q\_SLOTS:

void **addTask**();

void **editTask**();

void **deleteTask**();

void **on\_calendarWidget\_clicked**(const QDate &date);

private:

void **initActions**();

void **showTasks**();

void **showTask**(Task\* task, int index);

private:

Ui::MainWindow \*ui = nullptr;

Calendar\* m\_calendar = nullptr;

// Выбранная на календаре дата:

// в списке отображаются задачи только на этот день

QDate m\_dateFilter;

};

#endif // MAINWINDOW\_H

mainwindow.cpp

#include "mainwindow.h"

#include "ui\_mainwindow.h"

#include "taskdlg.h"

#include "task.h"

#include "birthday.h"

#include "meeting.h"

#include "calendar.h"

#include <QDebug>

#include <QImage>

//#include <QToolButton>

MainWindow::**MainWindow**(QWidget \*parent) :

QMainWindow(parent),

ui(new Ui::MainWindow)

{

ui->setupUi(this);

m\_calendar = new Calendar();

m\_dateFilter = QDate::currentDate();

QStringList List;

initActions();

setWindowTitle(tr("Личный календарь"));

setMinimumSize(500, 350);

}

MainWindow::~***MainWindow***()

{

delete ui;

}

// реализация слотов

void MainWindow::**addTask**()

{

TaskDlg dlg;

dlg.setModal(true);

if(dlg.*exec*() && dlg.task()) {

// добавляем задачу в календарь

m\_calendar->addTask(dlg.task());

showTask(dlg.task(), m\_calendar->taskCount() - 1);

}

}

void MainWindow::**editTask**()

{

QListWidgetItem\* item = ui->lstTasks->currentItem();

if(!item) {

return;

}

int index = item->*data*(Qt::UserRole).toInt();

const Task\* task = m\_calendar->task(index);

if(!task) {

qDebug() << "inconsistent program state: index = " << index;

return;

}

TaskDlg\* dlg = 0;

if(auto meeting = dynamic\_cast<const Meeting\*>(task)) {

dlg = new TaskDlg(meeting->when(), meeting->name(),

meeting->who(), meeting->where());

} else if(auto birthday = dynamic\_cast<const Birthday\*>(task)) {

dlg = new TaskDlg(birthday->when(), birthday->name(),

birthday->who(), birthday->presents());

} else if(auto call = dynamic\_cast<const Interaction\*>(task)) {

dlg = new TaskDlg(call->when(), call->name(),

call->who());

} else { // базовый класс Task

dlg = new TaskDlg(task->when(), task->name());

}

if(dlg) {

dlg->setModal(true);

if(dlg->*exec*() && dlg->task()) {

// обновлем параметры задачи в календаре

m\_calendar->setTask(index, dlg->task());

// сохраняем в этом элементе индекс задачи

item->*setData*(Qt::UserRole, m\_calendar->taskCount() - 1);

// добавляем элемент в список

item->setText(dlg->task()->*toString*());

}

delete dlg;

dlg = 0;

}

}

void MainWindow::**deleteTask**()

{

int row = ui->lstTasks->currentRow();

QListWidgetItem\* item = ui->lstTasks->currentItem();

if(!item) {

return;

}

int index = item->*data*(Qt::UserRole).toInt();

const Task\* task = m\_calendar->task(index);

if(!task) {

qDebug() << "inconsistent program state: index = " << index;

return;

}

m\_calendar->delTask(index);

showTasks();

if(ui->lstTasks->count()) {

if(ui->lstTasks->count() == row)

--row;

ui->lstTasks->setCurrentRow(row);

}

}

void MainWindow::**on\_calendarWidget\_clicked**(const QDate &date)

{

m\_dateFilter = date;

showTasks();

}

// реализация методов

void MainWindow::**initActions**()

{

// зададим стандартные горячие клавиши для действий

ui->actNew->setShortcuts(QKeySequence::New);

ui->actNew->setStatusTip("Создать новую задачу");

//const QImage image("add.png");

//ui->actNew->setIcon(QPixmap::fromImage(image));

ui->actNew->setIcon(QIcon("img/add.png"));

connect(ui->actNew, &QAction::triggered, this, &MainWindow::addTask);

ui->actEdit->setStatusTip("Редактировать выбранную задачу");

ui->actEdit->setIcon(QIcon("img/edit.png"));

connect(ui->actEdit, &QAction::triggered, this, &MainWindow::editTask);

ui->actDelete->setShortcuts(QKeySequence::Delete);

ui->actDelete->setStatusTip("Удалить выбранную задачу");

ui->actDelete->setIcon(QIcon("img/del.png"));

connect(ui->actDelete, &QAction::triggered, this, &MainWindow::deleteTask);

ui->actExit->setShortcuts(QKeySequence::Quit);

ui->actExit->setStatusTip("Завершение работы");

connect(ui->actExit, &QAction::triggered, this, &MainWindow::close);

// Toolbar

//QToolButton\* btnNew = new QToolButton;

//btnNew->setDefaultAction(ui->actNew);

//ui->mainToolBar->addWidget(btnNew);

ui->mainToolBar->addAction(ui->actNew);

ui->mainToolBar->addAction(ui->actEdit);

ui->mainToolBar->addAction(ui->actDelete);

}

void MainWindow::**showTasks**()

{

ui->lstTasks->clear();

int taskCount = m\_calendar->taskCount();

for(int i = 0; i < taskCount; ++i) {

Task\* task = m\_calendar->task(i);

if(task->when().date() == m\_dateFilter) {

showTask(task, i);

}

}

}

void MainWindow::**showTask**(Task\* task, int index)

{

const char\* imgFileName = "";

// создаём элемент визуального списка

if(dynamic\_cast<const Meeting\*>(task)) {

imgFileName = "img/share.png";

} else if(dynamic\_cast<const Birthday\*>(task)) {

imgFileName = "img/shopping-cart.png";

} else if(dynamic\_cast<const Interaction\*>(task)) {

imgFileName = "img/phone-call.png";

} else { // базовый класс Task

imgFileName = "img/star.png";

}

auto item = new QListWidgetItem(QIcon(imgFileName), task->*toString*());

// сохраняем в этом элементе индекс задачи

item->*setData*(Qt::UserRole, index);

// добавляем элемент в список

ui->lstTasks->addItem(item);

}

taskdlg.h

#ifndef TASKDLG\_H

#define TASKDLG\_H

#include <QDialog>

#include "task.h"

namespace Ui {

class TaskDlg;

}

class TaskDlg : public QDialog

{

Q\_OBJECT

public:

// для добавления

explicit TaskDlg(QWidget \*parent = 0);

// для редактирования дела (Task)

explicit TaskDlg(const QDateTime& when, const QString& name, QWidget \*parent = 0);

// для редактирования звонка (Interaction)

explicit TaskDlg(const QDateTime& when, const QString& name, const QString& who, QWidget \*parent = 0);

// для редактирования встречи (Meeting)

explicit TaskDlg(const QDateTime& when, const QString& name, const QString& who, const QString& place, QWidget \*parent = 0);

// для редактирования ДР (Birthday)

explicit TaskDlg(const QDateTime& when, const QString& name, const QString& who, const QList<QString>& presents, QWidget \*parent = 0);

~***TaskDlg***();

Task\* **task**() const {

return m\_task;

}

private slots:

void **on\_cmdAdd\_clicked**();

void **on\_cmdDelete\_clicked**();

void **on\_optTask\_clicked**();

void **on\_optMeeting\_clicked**();

void **on\_optCall\_clicked**();

void **on\_optBirthday\_clicked**();

void **on\_buttonBox\_accepted**();

private:

Ui::TaskDlg \*ui;

Task\* m\_task = 0;

TaskType m\_taskType = ttCommon;

private:

void **init**();

};

#endif // TASKDLG\_H

taskdlg.cpp

#include "taskdlg.h"

#include "ui\_taskdlg.h"

#include "calendar.h"

#include "birthday.h"

#include "meeting.h"

#include <QDebug>

#include <QInputDialog>

#include <QDateTime>

void TaskDlg::**init**()

{

ui->setupUi(this);

}

// для добавления любой задачи

TaskDlg::**TaskDlg**(QWidget \*parent) :

QDialog(parent),

ui(new Ui::TaskDlg)

{

init();

ui->optTask->animateClick();

ui->dateTimeEdit->setDateTime(QDateTime::currentDateTime());

}

// для редактирования дела (Task)

TaskDlg::**TaskDlg**(const QDateTime& when, const QString& name, QWidget \*parent) :

QDialog(parent),

ui(new Ui::TaskDlg)

{

init();

ui->optTask->animateClick();

ui->txtName->setText(name);

ui->dateTimeEdit->setDateTime(when);

}

// для редактирования звонка (Interaction)

TaskDlg::**TaskDlg**(const QDateTime& when, const QString& name, const QString& who, QWidget \*parent) :

QDialog(parent),

ui(new Ui::TaskDlg)

{

init();

ui->optCall->animateClick();

ui->txtName->setText(name);

ui->txtWho->setText(who);

ui->dateTimeEdit->setDateTime(when);

}

// для редактирования встречи (Meeting)

TaskDlg::**TaskDlg**(const QDateTime& when, const QString& name, const QString& who, const QString& place, QWidget \*parent) :

QDialog(parent),

ui(new Ui::TaskDlg)

{

init();

ui->optMeeting->animateClick();

ui->txtName->setText(name);

ui->txtWho->setText(who);

ui->txtWhere->setText(place);

ui->dateTimeEdit->setDateTime(when);

}

// для редактирования ДР (Birthday)

TaskDlg::**TaskDlg**(const QDateTime& when, const QString& name, const QString& who, const QList<QString>& presents, QWidget \*parent) :

QDialog(parent),

ui(new Ui::TaskDlg)

{

init();

ui->optBirthday->animateClick();

ui->txtName->setText(name);

ui->txtWho->setText(who);

for(const auto& present : presents) {

ui->listWidget->addItem(present);

}

ui->dateTimeEdit->setDateTime(when);

}

TaskDlg::~***TaskDlg***()

{

qDebug() << "TaskDlg::destructor";

delete ui;

}

void TaskDlg::**on\_cmdAdd\_clicked**()

{

bool ok;

QString name = QInputDialog::getText(nullptr, "Новый подарок", "Подарок: ", QLineEdit::Normal, "", &ok);

if(ok && !name.isEmpty()) {

auto item = new QListWidgetItem(name);

ui->listWidget->addItem(item);

}

}

void TaskDlg::**on\_cmdDelete\_clicked**()

{

;

}

void TaskDlg::**on\_optTask\_clicked**()

{

qDebug() << "TASK";

m\_taskType = ttCommon;

ui->txtName->setText("Дело");

ui->gbxList->*setVisible*(false);

ui->txtWhere->setEnabled(false);

ui->lblWhere->setEnabled(false);

ui->txtWho->setEnabled(false);

ui->lblWho->setEnabled(false);

ui->lblWho->setText("С кем");

}

void TaskDlg::**on\_optMeeting\_clicked**()

{

qDebug() << "MEETING";

m\_taskType = ttMeeting;

ui->txtName->setText("Встреча");

ui->gbxList->*setVisible*(false);

ui->txtWhere->setEnabled(true);

ui->lblWhere->setEnabled(true);

ui->txtWho->setEnabled(true);

ui->lblWho->setEnabled(true);

ui->lblWho->setText("С кем");

}

void TaskDlg::**on\_optCall\_clicked**()

{

qDebug() << "CALL";

m\_taskType = ttCall;

ui->txtName->setText("Звонок");

ui->gbxList->*setVisible*(false);

ui->txtWhere->setEnabled(false);

ui->lblWhere->setEnabled(false);

ui->txtWho->setEnabled(true);

ui->lblWho->setEnabled(true);

ui->lblWho->setText("Кому");

}

void TaskDlg::**on\_optBirthday\_clicked**()

{

qDebug() << "BIRTHDAY";

m\_taskType = ttBirthday;

ui->txtName->setText("ДР");

ui->gbxList->*setVisible*(true);

ui->txtWhere->setEnabled(false);

ui->lblWhere->setEnabled(false);

ui->txtWho->setEnabled(true);

ui->lblWho->setEnabled(true);

ui->lblWho->setText("У кого");

}

void TaskDlg::**on\_buttonBox\_accepted**()

{

if(m\_task)

delete m\_task;

switch(m\_taskType) {

case ttCommon:

m\_task = new Task(ui->dateTimeEdit->dateTime(), ui->txtName->text());

break;

case ttMeeting:

m\_task = new Meeting(ui->dateTimeEdit->dateTime(), ui->txtName->text(),

ui->txtWho->text(), ui->txtWhere->text());

break;

case ttCall:

m\_task = new Interaction(ui->dateTimeEdit->dateTime(), ui->txtName->text(),

ui->txtWho->text());

break;

case ttBirthday:

m\_task = new Birthday(ui->dateTimeEdit->dateTime(), ui->txtName->text(),

ui->txtWho->text());

for(int i = 0; i < ui->listWidget->count(); ++i) {

static\_cast<Birthday\*>(m\_task)->addPresent(ui->listWidget->item(i)->text());

}

break;

default:

m\_task = 0;

}

}