GeekBrains

C++: разработка программ с графическим интерфейсом на Qt

Работа с графикой и текстом в Qt

Знакомство с QPainter. Создание интерактивного графического приложения с помощью Graphics View Framework

Контекст рисования QPainter

Класс QPainter

Класс **QPainter** позволяет управлять изменением цвета пикселей объекта для отрисовки линий и геометрических фигур. В предыдущих уроках этот класс уже использовался для отрисовки результата работы OpenGL и при отрисовке содержимого для печати на принтере. Этот класс также является низкоуровневым классом оформления виджетов. С помощью таблицы стилей можно поменять цвет фона или шрифт текста, а с помощью **QPainter —** полностью перерисовать виджет или нарисовать собственный. Класс QPainter объявляют так: ***painter = new QPainter(виджет, в области которого будет происходить отрисовка)**. Отрисовка заключена между методами класса — **begin()** и **end()**. Методы отрисовки начинаются с английского слова draw: например, **drawLine**, **drawText**.

Функция **isActive()** указывает, активен ли режим отрисовки. Отрисовка активируется функцией **begin()** и конструктором, который принимает аргумент **QPaintDevice**. Функция **end()** и деструктор деактивируют ее. **QPainter** вместе с классами **QPaintDevice** и **QPaintEngine** формирует основу для системы рисования в Qt. Класс **QPainter** используется для выполнения операций рисования. **QPaintDevice** представляет собой устройство, на котором можно рисовать с помощью объекта класса **QPainter**. **QPaintEngine** предоставляет интерфейс, который используется объектом класса **QPainter** для рисования на разных типах устройств (например, **QWidget**, **QPrinter**, **Graphics View Framework**). Если режим отрисовки активен, **device()** возвращает устройство рисования, на котором происходит отрисовка, а **paintEngine()** возвращает механизм рисования, на котором в данный момент работает **QPainter**.

Системы координат. Точка, линия, прямоугольник, полигон

Системы координат

Начало координат находится в верхнем левом углу клиентской области.

Точка, линия, прямоугольник, полигон

Точка

Метод класса **QPainter**. У этого метода три перегрузки. Нарисовать точку можно, например, передав два аргумента — координаты по оси X и по оси Y. Также в качестве аргумента можно передать структуру — в этом случае у метода будут две перегрузки: структуры **QPoint** и **QPointF**.

```
QPainter painter(this);
   painter.drawPoint(50, 50);  // x, y
   QPoint thPoint(50, 50);
   painter.drawPoint(thPoint); // по объекту точки
   QPointF thPointf(50., 50.);
   painter.drawPoint(thPointf);
   this->render(this);
```

Линия

Для отрисовки линии используется метод **drawLine**, для задания координат можно использовать четыре целочисленных значения координат начала и конца линии или использовать объекты точек (**QPoint** и **QPointF**) и объект линии (**QLine**, **QLineF**).

```
QPainter painter(this);
   painter.drawLine(0,0, 100, 100); // x1, y1, x2, y2
   QPoint p1 (0,0);
   QPoint p2 (100,100);
   painter.drawLine(p1,p2);         // по двум точкам
   QLineF line(0,0,100,100);
   painter.drawLine(line);          // по объекту "линия"
   painter.end();
   this->render(this);
```

Прямоугольник

Для отрисовки прямоугольника используется метод **drawRect**, у которого может быть как четыре целочисленных аргумента (координаты левого верхнего угла прямоугольника, его ширина и высота), так и один аргумент в виде специального объекта координат **QRect** или **QRectF**.

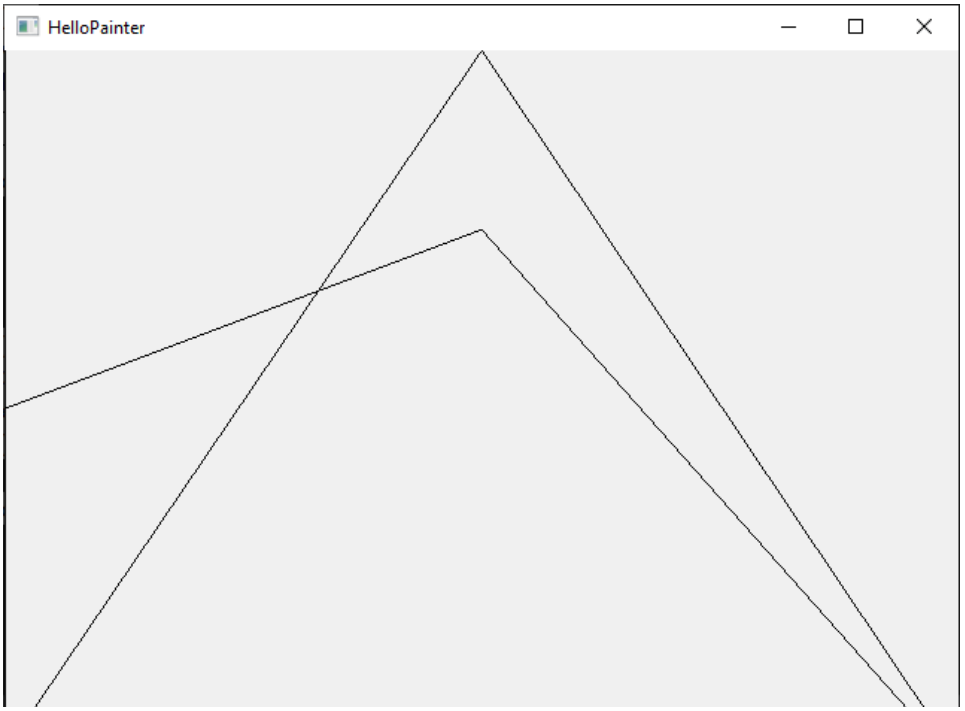
```
QPainter painter(this);
   painter.drawRect(QRect(0,0,100,100));
   painter.drawRect(0, 20, 50, 50);
   painter.drawRect(QRectF(100, 120, 200, 200));
   painter.end();
   this->render(this);
```

Полигон

Для создания более сложных геометрических фигур используется метод **drawPolygon**, который отрисовывает линии по заданному массиву точек:

```
QPainter painter(this);
   QPolygon polygon;                          // через объект QPolygon
   polygon << QPoint(0, height() -- 10);
   polygon << QPoint(width() >> 1, 0);
   polygon << QPoint(width(), height() -- 5);
   painter.drawPolygon(polygon);

   //Массив точек
   QPointF points[] = {
       QPointF(0,0),                          // 0
       QPointF(0,height() >> 1),              // 1
       QPointF(width() >> 1, height() >> 2), // 2
       QPointF(width(),height()),             // 3
       QPointF(0,height()),                   // 4
```

```
    };
    painter.drawPolygon(points, sizeof(points) / sizeof(points[0]));
    painter.end();
    this->render(this);
```



Окружность

Метод отрисовки окружности (эллипса) — **drawEllipse**, аргументы можно указать как для прямоугольника, но можно и задать координаты центра с радиусами по осям x и y.

```
// Задаем координаты x и y верхнего угла, ширину и высоту
painter.drawEllipse(50, 50, 50, 50);
// Задаем прямоугольную область
painter.drawEllipse(QRectF(50.0, 50.0, 50.0, 50.0));
painter.drawEllipse(QRect(50, 50, 50, 50));
// Задаем центр через класс точки и радиусы по осям x и y
painter.drawEllipse(QPoint(50,50), 50, 50);
painter.drawEllipse(QPointF(50.0,50.0), 50.0, 50.0);
```

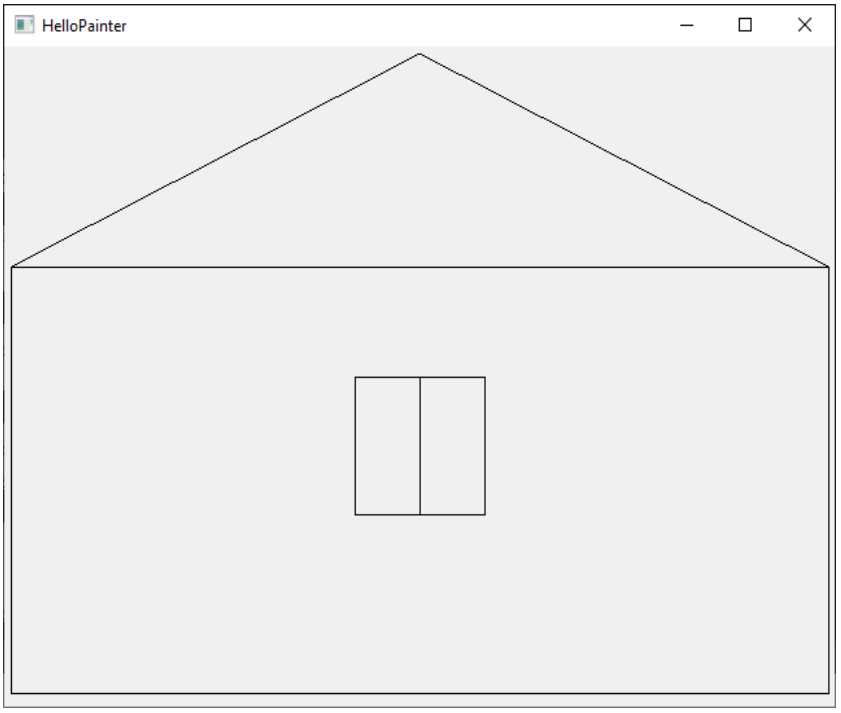
Нарисуем дом с помощью примитивов. Создадим новый проект без создания формы с родительским классом **QWidget**. Напишем программу, которая будет, используя основные примитивы, рисовать дом.

```
#ifndef WIDGET_H
#define WIDGET_H

#include <QWidget>
```

```
class Widget : public QWidget
{
   Q_OBJECT

public:
   Widget(QWidget *parent = 0);
   ~Widget();
protected:
   void paintEvent(QPaintEvent *event);
};

#endif // WIDGET_H
```

В примере отрисовка происходит при наступлении соответствующих событий (**paintEvent**). Для этого используется класс **QPainter**.

```
#include "widget.h"
#include <QPainter>
#include <QPolygon>
Widget::Widget(QWidget *parent)
   : QWidget(parent)
{
}

Widget::~Widget()
{

}

void Widget::paintEvent(QPaintEvent *event)
{
   QPainter painter(this);
   int h = height() / 3;
   // Дом
   painter.drawRect(5, h, width() -- 10, height() -- h -- 10);
   int w = width() >> 1;
   // Крыша
   QPolygon polygon;
   polygon << QPoint(w, 5);
   polygon << QPoint(5, h);
   polygon << QPoint(width() -- 5, h);
   painter.drawPolygon(polygon);

   // Окошко
   h = height() >> 1;
   painter.drawRect(w -- 50, h, 100, 100);
   painter.drawLine(w, h, w, h + 100);
   painter.end();
   this->render(this);
}
```

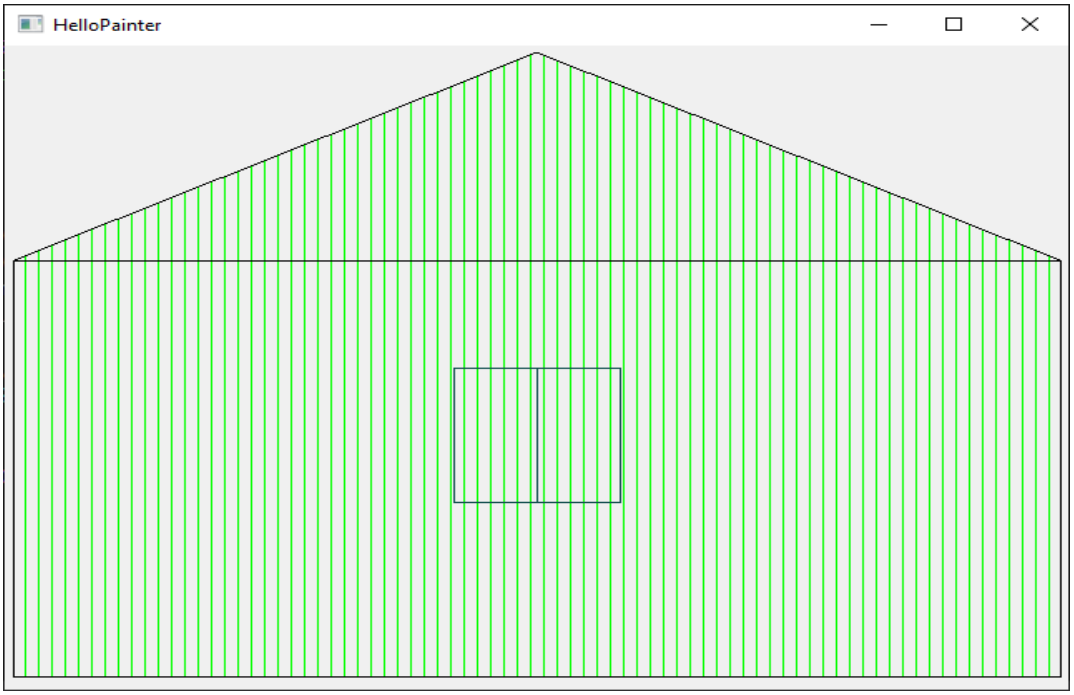


Перья и кисти. Цветовые модели. Градиенты

Перья и кисти

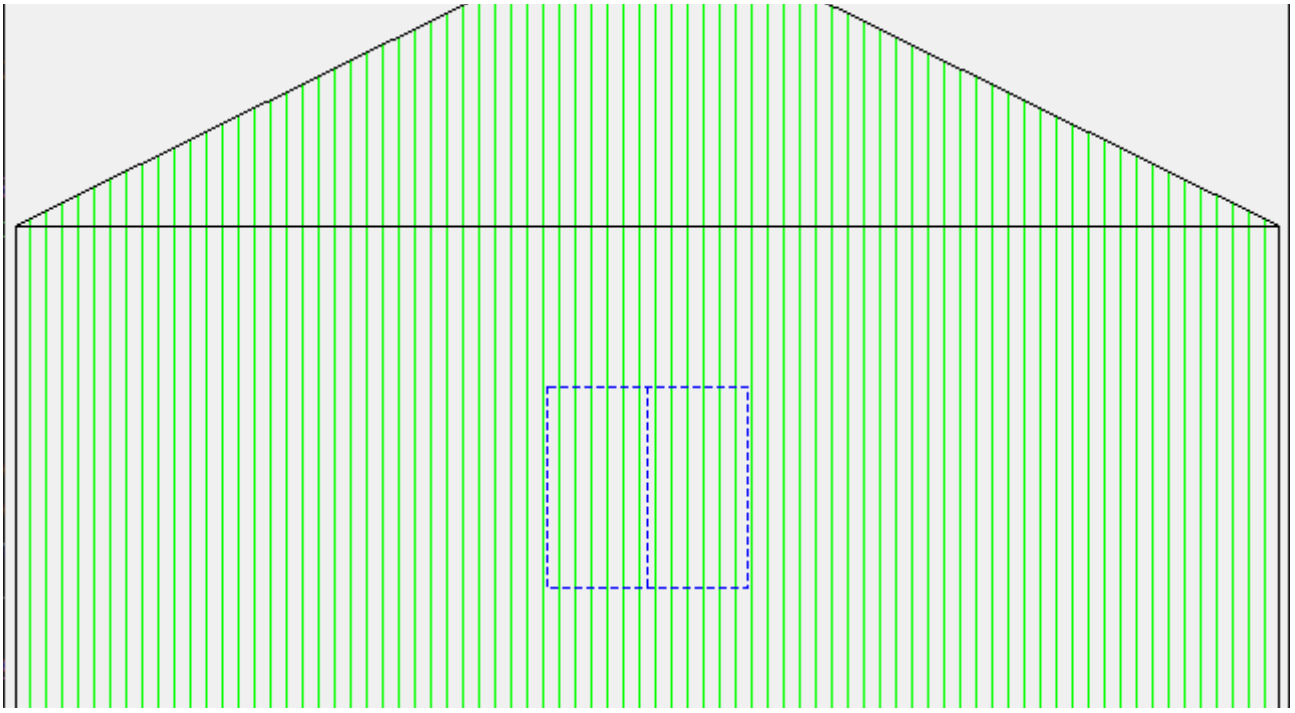
Для отрисовки содержимого требуется изменять цвет как линий, так и заливки фигуры. Для изменения цвета линии используют класс **QPen**, а за цвет заливки отвечает класс **QBrush**. Окрашивать можно прямоугольник, эллипс и произвольный полигон. Классы **QPen** и **QBrush** содержат метод установки стиля и цвета отрисовки и заливки соответственно.

```
// Дом
  QBrush br(QColor(5, 255, 0));
  br.setStyle(Qt::BrushStyle::VerPattern);
  painter.setBrush(br);
  painter.drawRect(5, h, width() -- 10, height() -- h -- 10);
  int w = width() >> 1;
```



Похожим образом устанавливаются и стиль и цвет для линий.

```
// Окошко
  QPen pen(QColor(0, 0, 250));
  pen.setStyle(Qt::PenStyle::DashLine);
  painter.setPen(pen);
  h = height() >> 1;
  painter.drawRect(w -- 50, h, 100, 100);
  painter.drawLine(w, h, w, h + 100);
  painter.end();
  this->render(this);
```



Graphics View Framework. Сцена и представление

Graphics View Framework

Graphics View Framework позволяет работать с 2D-графикой, предоставляя такие возможности, как поворот, масштабирование графического содержимого, широкий набор возможностей по созданию интерактивных приложений. Graphics View Framework впервые появился в Qt 4.2, заменив его предшественника, QCanvas. Виджет Graphics View Framework использует для отображения класс **QGraphicsView** и модель отображения **QGraphicsScene**. Эта связка объектов похожа на **QTreeView**, **QTableView** и **QListView** с **QModel**, но для фреймворка отображения графики.

Основные классы Graphics View Framework:

Класс	Описание
QGraphicsEffect	Базовый класс для всех графических эффектов
QGraphicsAnchor	Связка между двумя элементами в QGraphicsAnchorLayout
QGraphicsAnchorLayout	Макет, в котором можно связать виджеты вместе в графическом представлении
QGraphicsGridLayout	Сетка для управления виджетами в графическом представлении
QAbstractGraphicsShapeItem	Общая база для всех элементов

QGraphicsEllipseItem, QGraphicsLineItem, QGraphicsRectItem, QGraphicsSimpleTextItem	Элементы — эллипс, линия, прямоугольник, текст
QGraphicsItem	Базовый класс для всех графических элементов в QGraphicsScene

Классы событий:

- QGraphicsSceneContextMenuEvent
- QGraphicsSceneDragDropEvent
- QGraphicsSceneEvent
- QGraphicsSceneHelpEvent
- QGraphicsSceneHoverEvent
- QGraphicsSceneMouseEvent
- QGraphicsSceneMoveEvent
- QGraphicsSceneResizeEvent
- QGraphicsSceneWheelEvent

События здесь практически такие же, как и у остальных виджетов: вызов контекстного меню, изменение размера и события мыши и клавиатуры.

Сцена и представление

Создадим проект отрисовки простой сцены через фреймворк Graphics View Framework. Создадим новый проект с базовым классом **QMainWidget**, а после этого поменяем его на базовый класс **QGraphicsItem**, чтобы создать собственный визуальный объект отрисовки:

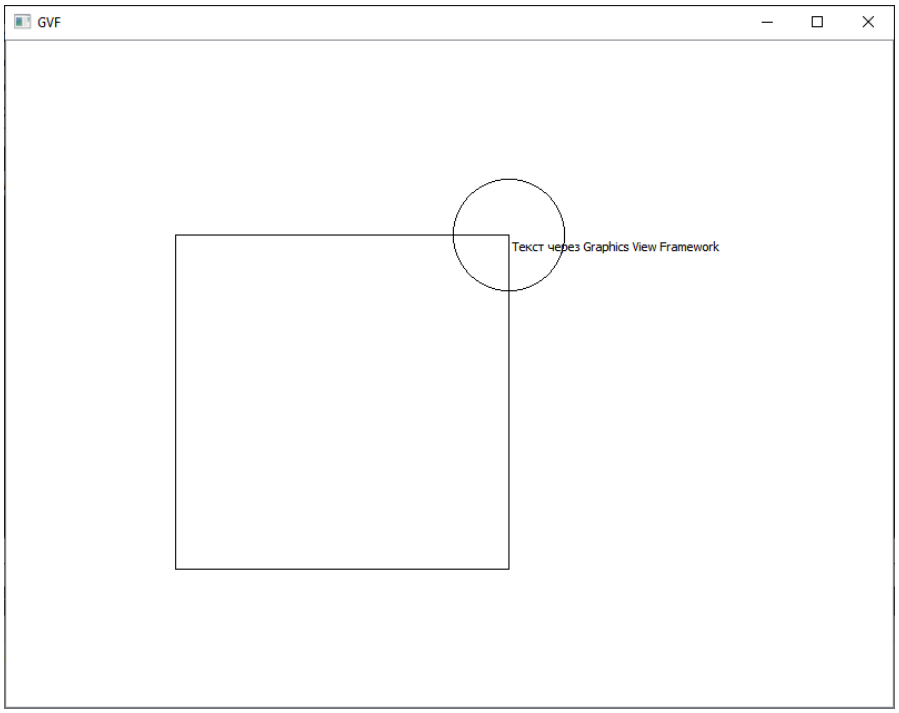
```
#ifndef MAINWINDOW_H
#define MAINWINDOW_H

#include <QGraphicsScene>
#include <QGraphicsView>

class MainWindow : public QGraphicsView
{
   Q_OBJECT

public:
   MainWindow(QWidget *parent = 0);
   ~MainWindow();
private:
   QGraphicsView *view;
   QGraphicsScene *scence;
protected:
private slots:
```

```
};

#endif // MAINWINDOW_H
```

Теперь нарисуем прямоугольник, текст и эллипс:

```
#include "mainwindow.h"
#include <QMessageBox>

MainWindow::MainWindow(QWidget *parent)
    : QGraphicsView(parent)
{
    scence = new QGraphicsScene(this);                      // Новая сцена
    setScene(scence);
    scence->addRect(-300,0,300,300);                        // Добавляем
                                                            // прямоугольник
    scence->addText("Текст через Graphics View Framework"); // Добавляем текст
    scence->addEllipse(-50, -50, 100, 100);
}
MainWindow::~MainWindow()
{

}
```

В итоге мы получим следующие приложение:



Как можно заметить, начало координат располагается в центре виджета. Шаг сетки — один пиксель. В Graphics View Framework для эллипса указываются координаты центра эллипса и радиусы по осям X и Y. Другой способ — отрисовка с помощью класса **QPainter (**пример ниже**)**.

Добавим интерактивности приложению при помощи объекта **QGraphicsItem.** Создадим новый класс с наследованием от базового класса **QObject**:

```
#ifndef BLOCKSCHEME_H
#define BLOCKSCHEME_H

#include <QObject>
#include <QGraphicsItem>
#include <QContextMenuEvent>
#include <QBrush>


class BlockScheme : public QObject, public QGraphicsItem
{
   Q_OBJECT
   Q_PROPERTY(QBrush brush)
public:
   explicit BlockScheme(QObject *parent = nullptr);
   void setBrush(QBrush brush) {this->brush = brush; emit reDraw();}
signals:
   void reDraw();
   void dblClick();
public slots:
private:
   void paint(QPainter *painter, const QStyleOptionGraphicsItem *option, QWidget
*widget);
   QRectF boundingRect() const;
private:
   enum Geometry {RECTANGLE, ELLIPS};
   int x, y;
   Geometry geometry;
   QPoint bpoint;
   bool moving;
   QBrush brush;
protected:
   void mousePressEvent(QGraphicsSceneMouseEvent *event) override;
   void mouseReleaseEvent(QGraphicsSceneMouseEvent *event) override;
   void mouseMoveEvent(QGraphicsSceneMouseEvent *event) override;
   void mouseDoubleClickEvent(QGraphicsSceneMouseEvent *event) override;
};

#endif // BLOCKSCHEME_H
```

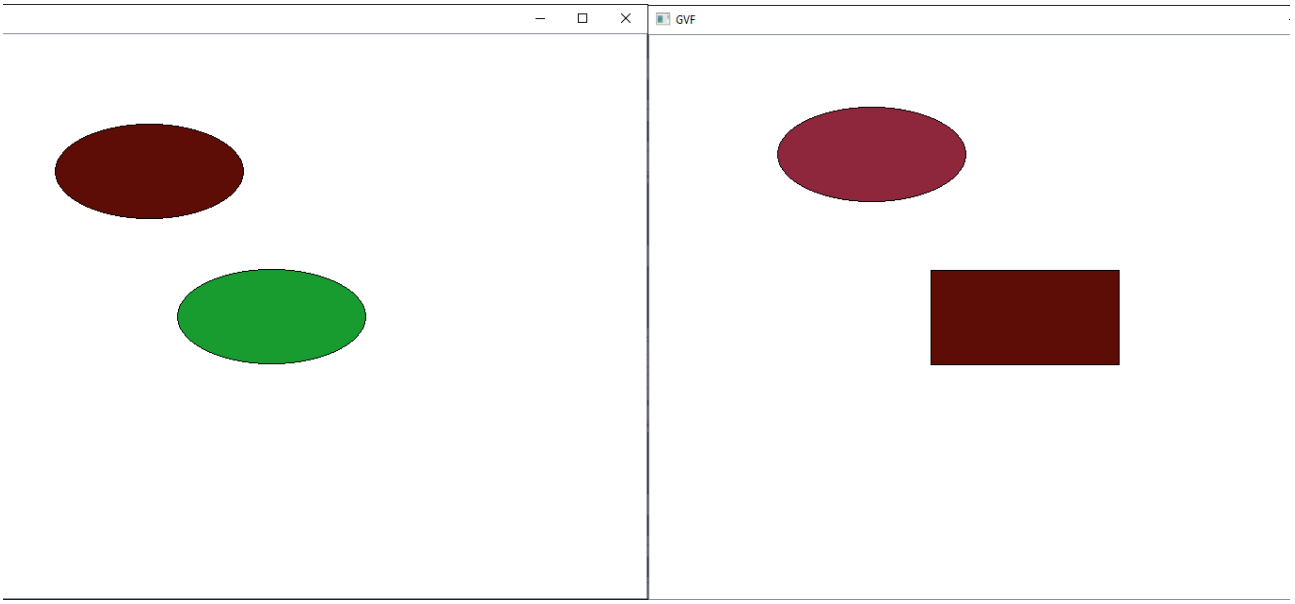
Этот класс наследует свойства и методы от двух классов фреймворка Qt. Добавляем переопределение следующих событий: нажатие кнопки мыши, перемещение и двойное нажатие

кнопки мыши. Когда нажата левая кнопка мыши, объект становится доступен для перемещения, а правой мы превращаем фигуру из эллипса в прямоугольник и обратно.

```
#include "blockscheme.h"
#include <QPainter>
#include <QGraphicsSceneMouseEvent>

BlockScheme::BlockScheme(QObject *parent) : QObject(parent), QGraphicsItem()
{
   x = 0;
   y = 0;
   brush.setColor(QColor(rand() % 256, rand() % 256, rand() % 256));
   brush.setStyle(Qt::BrushStyle::SolidPattern); // Полностью закрашивать
   geometry = Geometry::ELLIPS;                  // По умолчанию - эллипс
   setPos(0,0);
   moving = false;
}

void BlockScheme::paint(QPainter *painter, const QStyleOptionGraphicsItem
*option, QWidget *widget)
{
   painter->setBrush(brush);
   if (geometry == Geometry::ELLIPS) painter->drawEllipse(x, y, 200, 100);
   if (geometry == Geometry::RECTANGLE) painter->drawRect(x, y, 200, 100);
   Q_UNUSED(option)
   Q_UNUSED(widget)
}

QRectF BlockScheme::boundingRect() const          // Обязателен для
                                                  // переопределения
{
   return QRectF(x, y, 200, 100);                 // Текущие координаты
}

void BlockScheme::mousePressEvent(QGraphicsSceneMouseEvent *event)
{
   if (event->button() == Qt::LeftButton)         // Левая кнопка, режим
                                                  // перемещения
   {
       moving = true;                             // Флаг активности
                                                  // перемещения
       bpoint = event->pos().toPoint();           // Запоминаем начальные
                                                  // координаты мыши
   }
   if (event->button() == Qt::RightButton)        // Правая кнопка - меняем вид
                                                  // фигуры
   {
       if (geometry == Geometry::ELLIPS)
       {
           geometry = Geometry::RECTANGLE;
       } else geometry = Geometry::ELLIPS;
       emit reDraw();                             // переотрисовка
```

```
    }
}

void BlockScheme::mouseReleaseEvent(QGraphicsSceneMouseEvent *event)
{
    if (event->button() == Qt::LeftButton)
    {
        moving = false;                                 // Снимаем флаг на
                                                        // перемещение
        emit reDraw();
    }
}

void BlockScheme::mouseMoveEvent(QGraphicsSceneMouseEvent *event)
{
    if (moving)                                         // Если активен флаг
                                                        // перемещения
    {
// Вычисляем вектор смещения
        QPoint p = event->pos().toPoint() -- bpoint;
        x += p.x();
        y += p.y();
        bpoint = event->pos().toPoint();                // Запоминаем новую позицию
                                                        // курсора
        emit reDraw();                                  // Переотрисовываем
    }
}

void BlockScheme::mouseDoubleClickEvent(QGraphicsSceneMouseEvent *event)
{
    if (event->button() == Qt::LeftButton)
    {
        emit dblClick();                                 // Отправляем сигнал
                                                         // о двойном клике
    }
}
```

Кроме того, используя методы класса QPainter, можно добавить текст или изображения в данный объект. С помощью этого модуля удобно создавать картографические, схемотехнические (например, для разводки печатных плат) и другие программы, где необходимо графическое отображение элементов.



Текст с элементами форматирования. Шрифты

Текст с элементами форматирования

Многие форматы текстовых документов позволяют форматировать текст (изменять, например, начертание шрифта, цвет и размер символов). Примером форматированного текста может послужить документ, который вы сейчас читаете. В виджете **QTextEdit** можно форматировать текст с помощью HTML-разметки либо через объекты **QTextCharFormat** и **QTextFormat**. Для применения форматирования к выделенному фрагменту текста понадобится объект **QTextCursor**. Создадим новый проект, в котором будет производиться форматирование текста, главное окно которого будет на базовом классе **QMainWindow**. Отредактируем заголовочный файл класса главного окна.

```
#ifndef MAINWINDOW_H
#define MAINWINDOW_H

#include <QMainWindow>
#include <QTextEdit>

class MainWindow : public QMainWindow
{
   Q_OBJECT

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

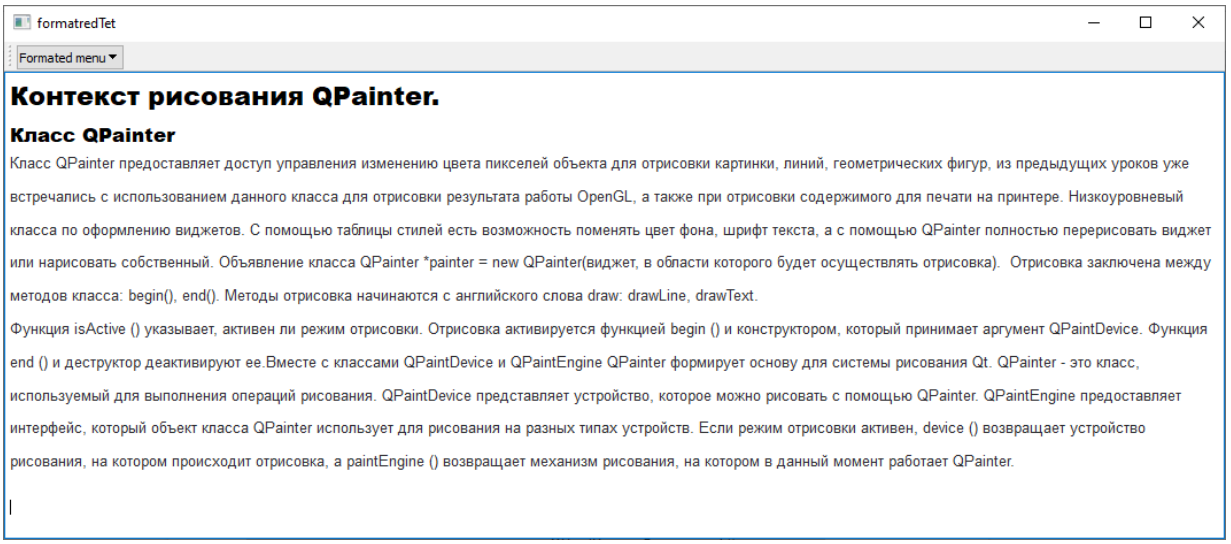
```
private:
   QTextEdit *tEdit;
private slots:
   void randomSizeOfFont(); // Случайный цвет шрифта и фона
};

#endif // MAINWINDOW_H
```

Добавим кнопку с меню на панель инструментов. В меню кнопки будет одна команда, которая будет случайным образом выставлять цвет шрифта и цвет фона символа (маркер).

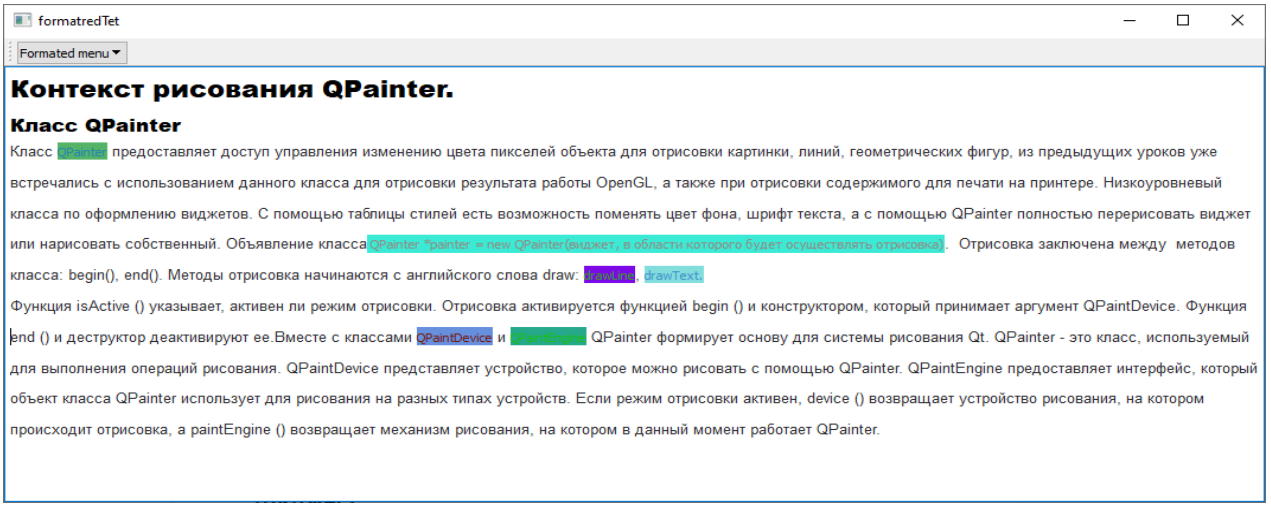
```
#include "mainwindow.h"
#include <QToolBar>
#include <QMenu>
#include <QPushButton>
#include <QTextDocumentFragment>
#include <QTextDocument>

MainWindow::MainWindow(QWidget *parent)
   : QMainWindow(parent)
{
   tEdit = new QTextEdit(this);
   setCentralWidget(tEdit);
   QToolBar *tBar = addToolBar("Formated");
   QMenu *menu = new QMenu(this);
   QAction *action = menu->addAction(tr("Random size of font"));
   connect(action, SIGNAL(triggered()), this, SLOT(randomSizeOfFont()));
   QPushButton *button = new QPushButton(tr("Formated menu"), this);
   tBar->addWidget(button);
   button->setMenu(menu);
   srand(clock());                          // Запускаем генератор
                                            // псевдослучайных чисел
                                            // от внутреннего таймера
}

MainWindow::~MainWindow()
{

}

void MainWindow::randomSizeOfFont()
{
   QTextCharFormat fmt;                      // Объект для форматирования
                                             // фрагмента текста
   fmt.setForeground(QBrush(QColor(rand() % 256,rand() % 256, rand() % 256)));
                                             // Цвет символа
                                             // (setForeground(QBrush))
   fmt.setBackground(QBrush(QColor(rand() % 256,rand() % 256, rand() % 256)));
   tEdit->textCursor().setCharFormat(fmt); // Задаем формат выделенного
                                             // участка текста
```

```
}
```

Запустив приложение, копируем произвольный текст и вставляем в текстовое поле приложения:



Выделяя произвольным образом текст, а затем выбирая пункт произвольного изменения форматирования, получим следующий результат:



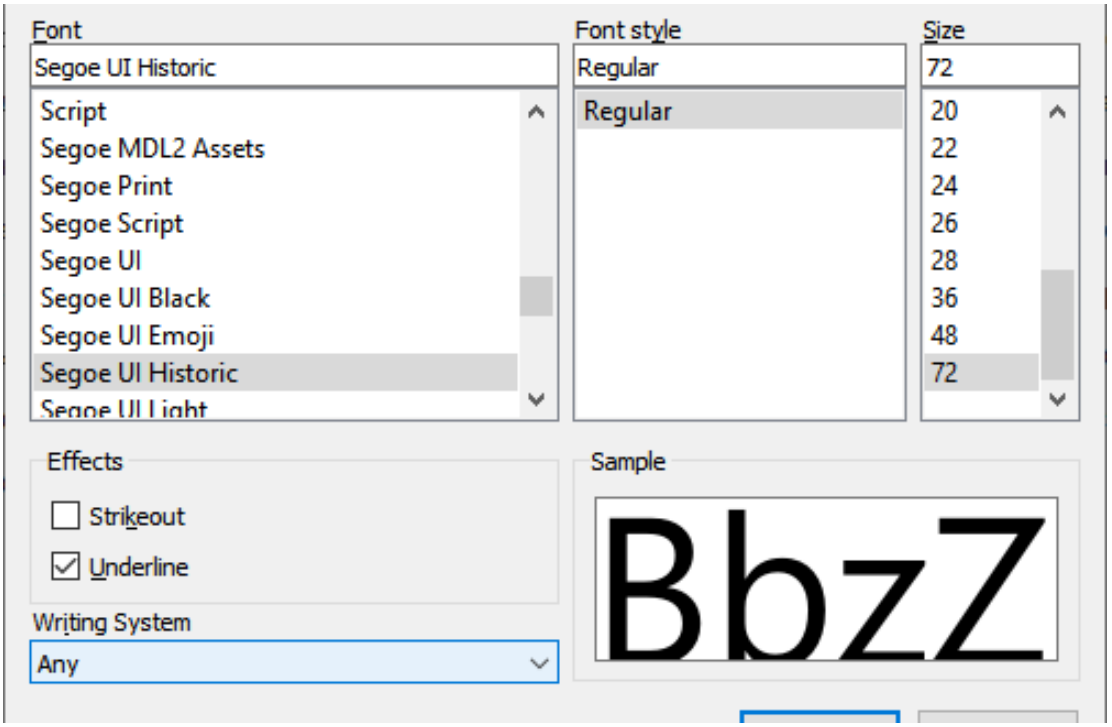
Шрифты

Класс **QFont** содержит настройки форматирования шрифта: имя, стиль, размер. Установить их можно с помощью диалогового окна **QFontDialog**. Доработаем проект: добавим пункт меню, настраивающий шрифт, и слот **setFont()**. Вот код слота:

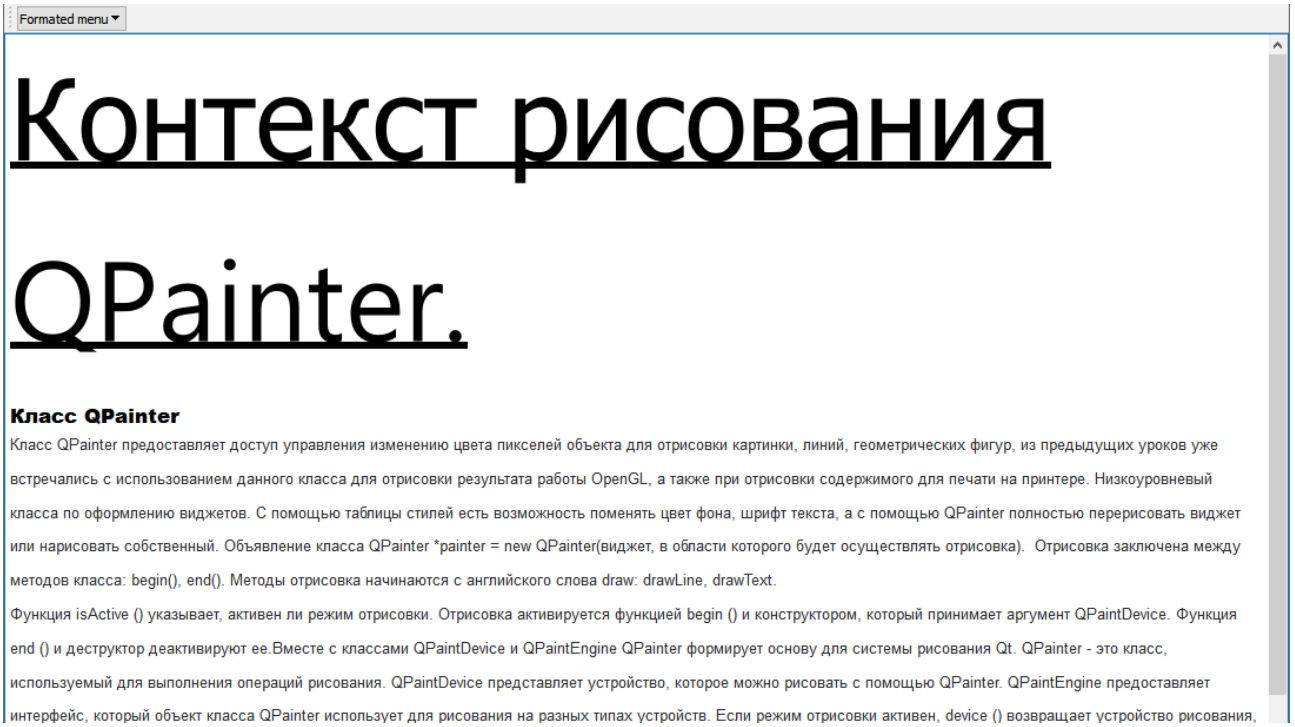
```
void MainWindow::setFont()
{
   QFont font = tEdit->textCursor().charFormat().font(); // получаем текущий
шрифт
   QFontDialog fntDlg(font,this);
```

```
    bool b[] = {true};
    font = fntDlg.getFont(b); // Запускаем диалог настройки шрифта
    if (b[0])                 // Если нажата кнопка OK, применяем новые
                              // настройки шрифта
    {
        QTextCharFormat fmt;
        fmt.setFont(font);
        tEdit->textCursor().setCharFormat(fmt);
    }

}
```

В полученном приложении копируем в текстовое поле произвольный текст (можно тот же самый, что и в предыдущем примере), выделяем верхнюю строку и выбираем пункт меню, открывающий настройки текста:



После применения настроек шрифта содержимое текстового поля программы будет выглядеть так:



У **QFont** есть методы для задания цвета (**setForeground(QBrush)**), стиля и остальных настроек шрифта. Подробнее о них можно прочитать здесь: https://doc.qt.io/qt-5/qfont.html.

Работа с HTML-разметкой. Вывод и сохранение

HTML (HyperText Markup Language — язык гипертекстовой разметки) не является языком программирования. Это язык разметки структуры веб-страниц. Текст и объекты страницы заключаются в теги. Добавим в код программы из предыдущего примера возможность использования гипертекста.

Работа с HTML-разметкой

Виджет QTextEdit поддерживает работу как с обычным текстом, так и с гипертекстом. Добавим два слота и два пункта меню: для преобразования текущего форматированного текста в HTML и HTML в форматированный текст.

```
// mainwindow.h
#ifndef MAINWINDOW_H
#define MAINWINDOW_H

#include <QMainWindow>
#include <QTextEdit>

class MainWindow : public QMainWindow
{
```

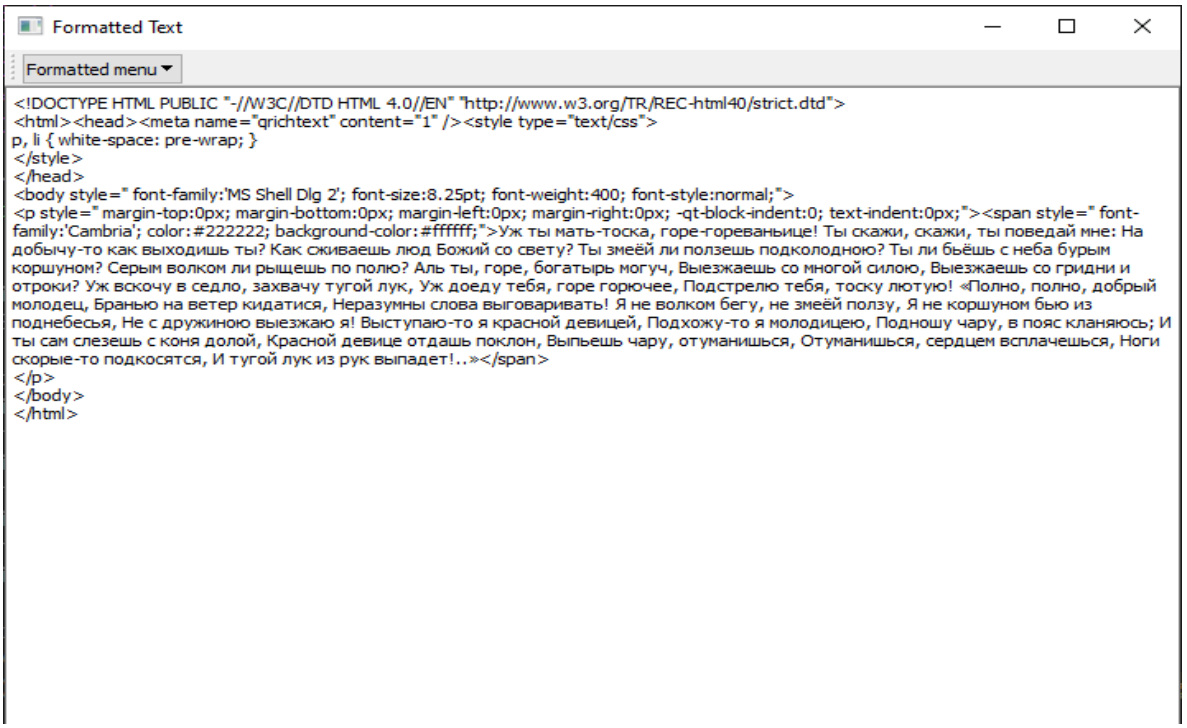
```
    Q_OBJECT

public:
    MainWindow(QWidget *parent = 0);
    ~MainWindow();
private:
    QTextEdit *tEdit;
private slots:
    void randomSizeOfFont();
    void setFont();
    void convertToHTML();
    void convertFromHTML();
};

#endif // MAINWINDOW_H


// mainwindow.cpp
MainWindow::MainWindow(QWidget *parent)
    : QMainWindow(parent)
{
    tEdit = new QTextEdit(this);
    setCentralWidget(tEdit);
    QToolBar *tBar = addToolBar("Formatted");
    QMenu *menu = new QMenu(this);
    QAction *action = menu->addAction(tr("Random size of font"));
    connect(action, SIGNAL(triggered()), this, SLOT(randomSizeOfFont()));
    action = menu->addAction(tr("Set font"));
    connect(action, SIGNAL(triggered()), this, SLOT(setFont()));
    action = menu->addAction(tr("Convert to HTML"));
    connect(action, SIGNAL(triggered()), this, SLOT(convertToHTML()));
    action = menu->addAction(tr("Convert from HTML"));
    connect(action, SIGNAL(triggered()), this, SLOT(convertFromHTML()));
    QPushButton *button = new QPushButton(tr("Formatted menu"), this);
    tBar->addWidget(button);
    button->setMenu(menu);
    srand(clock());
}
```

Теперь из виджета **QTextEdit** нужно получить текст либо гипертекст, а затем преобразовать соответственно в гипертекст или в форматированный текст в зависимости от выбранного пункта меню:

```
void MainWindow::convertToHTML()
{
    QString str = tEdit->document()->toHtml();      // Получаем HTML
    QString endTxt = "";
    QChar *chstr = str.data();
    for (int i = 0; i < str.length(); i)
    {
```

```
        int index = str.indexOf("</", i);
        if (index != -1)
        {
            index = str.indexOf(">", index);
            if (index != -1)
            {
                index++;
                endTxt.insert(endTxt.length(), &chstr[i], index - i);
                endTxt += "\n";
                i = index;
            }
            else
            {
                endTxt.insert(endTxt.length(), &chstr[i], str.length() - i);
                break;
            }
        }
        else
        {
            endTxt.insert(endTxt.length(), &chstr[i], str.length() - i);
            break;
        }
    }
    tEdit->document()->setPlainText(endTxt);        // Вставляем как обычный
                                                    // текст

}

void MainWindow::convertFromHTML()
{
    QString str = tEdit->document()->toPlainText(); // Получаем обычный текст,
                                                     // написанный в виджете
    tEdit->document()->setHtml(str);                 // Устанавливаем как HTML
}
```

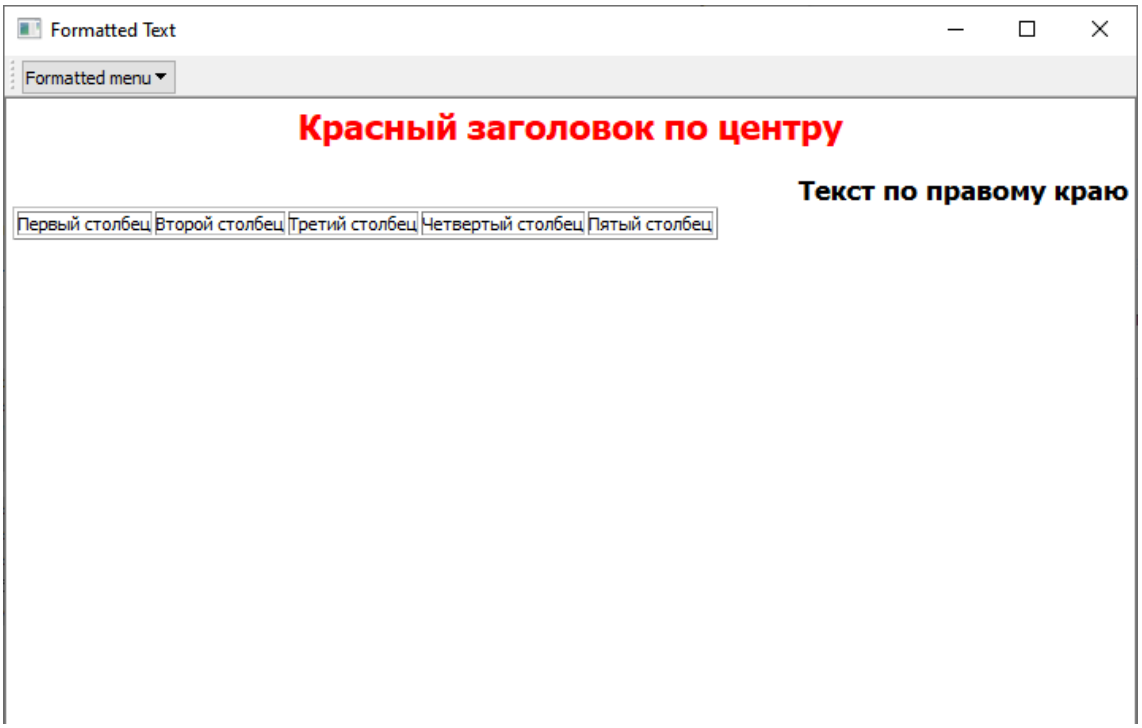
Вставив в приложение текст и преобразовав его в HTML, получим следующий результат:



Создадим простой HTML-документ:

```
<html>
<body>
<center><font color = "red"><h1>Красный заголовок по центру</h1></font></center>
<h2 align = "right">Текст по правому краю</h2>
<table border='1'>
<tr>
<td>Первый столбец</td>
<td>Второй столбец</td>
<td>Третий столбец</td>
<td>Четвертый столбец</td>
<td>Пятый столбец</td>
</tr>
</table>
</body>
</html>
```

Вставив этот текст в поле ввода приложения и преобразовав его из HTML в форматированный текст, получим следующий результат:



Вывод и сохранение

Сохранить форматированный текст можно как гипертекст, переведя его в формат HTML.

Вывод гипертекста на экран обеспечивают виджеты **QTextEdit** и **QWebView**: первый позволяет сохранить гипертекст в виде простого текстового файла, второй — в виде веб-архива mht.

Qt WebEngine. Реализация простого веб-браузера

WebEngine

Модуль WebEngine предоставляет набор классов для динамического рендеринга веб-контента в приложениях. Через qmake-файл (расширение файла *.pro) модуль подключается так: **QT+=webenginewidgets.**

Используя данный модуль, напишем простой веб-браузер, который сможет запоминать страницы в избранном и сохранять их в веб-архиве (*.mht),

Реализация простого веб-браузера

По умолчанию языком браузера будет английский. Добавим поддержку русского языка с помощью программы **Qt Linguist**. Скомпилированный файл локализации будет сохранен в ресурсах приложения. Создаем новый проект с базовым классом **QMainWindow** для главного окна без формы. Создадим файл ресурсов и файл локализации для русского языка:

```
QT       += core gui webenginewidgets


greaterThan(QT_MAJOR_VERSION, 4): QT += widgets

TARGET = MyWeb
TEMPLATE = app

DEFINES += QT_DEPRECATED_WARNINGS


#DEFINES += QT_DISABLE_DEPRECATED_BEFORE=0x060000    # disables all the APIs
deprecated before Qt 6.0.0

CONFIG += c++11

SOURCES += \
       main.cpp \
       mainwindow.cpp

HEADERS += \
       mainwindow.h

TRANSLATIONS += browser_ru.ts
CODECFRC += Utf-8
# Default rules for deployment.ф
qnx: target.path = /tmp/$${TARGET}/bin
else: unix:!android: target.path = /opt/$${TARGET}/bin
!isEmpty(target.path): INSTALLS += target

RESOURCES += \
   localization.qrc
```

Кнопку меню, строку навигации, а также кнопки перехода, добавления в избранное и меню избранного разместим на панели инструментов, а сам **QWebEngineView** разместим в качестве центрального виджета главного окна программы. Отредактируем заголовочный файл **mainwindow.h**:

```
#ifndef MAINWINDOW_H
#define MAINWINDOW_H

#include <QMainWindow>
#include <QLineEdit>
#include <QtWebEngineWidgets/QWebEngineView>
#include <QToolButton>
#include <QPushButton>
#include <QHash>
#include <QMenu>
class MainWindow : public QMainWindow
{
   Q_OBJECT
```

```
public:
    MainWindow(QWidget *parent = 0);
    ~MainWindow();
private:
    QLineEdit *navigateEdit;           // Строка навигации
    QWebEngineView *webView;           // Сам виджет веб-контента
    QToolButton *goButton;             // Кнопка для перехода на указанную
                                       // страницу
    QToolButton *addFavorite;          // Кнопка добавления страницы в избранное
    QPushButton *viewFavorite;         // Кнопка с выпадающим меню “Избранное”
    QPushButton *menuButton;           // Кнопка главного меню
    QMenu *favList;
    QHash<QString, QString>listFav;
private slots:
    void goToUrl();                    // Для перехода на указанный адрес
    void addToFavorite();              // Добавляем страницу в избранное
    void addToMenuFavorite(QString&); // Добавляем в пункт меню страницы ее
                                       // названия в списке избранного
    void openFavoritePage();           // Открываем избранную страницу
    void savePageAs();                 // Сохраняем содержимое виджета
                                       // QWebEngineView
private:
    void loadFavorites();              // При загрузке загружаем избранные
                                       // страницы в меню Избранное
    void addMainMenu();                // Создаем главное меню

};

#endif // MAINWINDOW_H
```

Начнем создание и компоновку виджетов в клиентской области окна:

```
#include "mainwindow.h"
#include <QToolBar>
#include <QFile>
#include <QFileDialog>
#include <QTextStream>
#include <QWebEnginePage>

MainWindow::MainWindow(QWidget *parent)
    : QMainWindow(parent)
{
    webView = new QWebEngineView(this);
    setCentralWidget(webView);
    QUrl q("https://geekbrains.ru/"); //Пусть домашней страницей будет сайт
                                      //IT-образовательного портала
    webView->setUrl(q);
    QToolBar *tool = addToolBar("Web Navigation");//создаем панель инструментов
    menuButton = new QPushButton(tr("Menu"), this);//создаем кнопку меню
    tool->addWidget(menuButton); // добавляем кнопку на панель инструментов
```

```
    navigateEdit = new QLineEdit(this);//создание строки навигации
    tool->addWidget(navigateEdit);
    goButton = new QToolButton(this);
    goButton->setText(tr("Go"));
    tool->addWidget(goButton);
    connect(goButton, SIGNAL(clicked()), this, SLOT(goToUrl()));
    addFavorite = new QToolButton(this);
    viewFavorite = new QPushButton(this);
    addFavorite->setText(tr("Add to favorite"));
    viewFavorite->setText(tr("Favorites"));
    tool->addWidget(addFavorite);
    tool->addWidget(viewFavorite);
    connect(addFavorite, SIGNAL(clicked()), this, SLOT(addToFavorite()));
    listFav.clear();
    favList = new QMenu(this);
    viewFavorite->setMenu(favList);
    loadFavorites();//загружаем избранные страницы в меню
    addMainMenu();//заполняем меню для кнопки menuButton
}
```

Теперь напишем код для слота перехода на указанную страницу в строке навигации:

```
void MainWindow::goToUrl()
{
    QString navTxt = navigateEdit->text();
    webView->setUrl(QUrl(navTxt));
}
```

Этот код достаточно прост: получаем строку из строки навигации, затем устанавливаем URL виджета **QWebEngineView**.

Добавим код для слота добавления страницы в избранное

```
void MainWindow::addToFavorite()
{
    QString txt = webView->url().toString();      // Получаем текущий адрес
                                                  // страницы
    if (txt.length() == 0) return;                // Проверяем, введен ли адрес

    QString title = webView->title();             // Получаем заголовок страницы

    QFile file("./favlist.bin");
    if (file.open(QIODevice::Append))
    {
        QByteArray ar = txt.toUtf8();             // Преобразуем строку в массив
                                                   // байтов
        int length = ar.length();                  // Определяем количество байт
        file.write((char*)&length, sizeof length);// Сохраняем информацию о
                                                   // количестве байт
```

```
        file.write(ar.data(),length);              // Сохраняем байты
                                                   // Повторяем ту же процедуру
                                                   // для заголовка
        ar = title.toUtf8();
        length = ar.length();
        file.write((char*)&length, sizeof length);
        file.write(ar.data(),length);
        file.close();
// Формируем уникальное имя избранной страницы для меню “Избранное”
        QString key = QString::number(listFav.count() + 1) + ". " + title;
        listFav[key] = txt;                         // Добавляем в контейнер по
                                                    // ключу заголовка
                                                    // содержимое – адрес
        addToMenuFavorite(key);                     // Добавляем в меню
    }
}
```

Храниться избранное будет в следующем виде: длина (в байтах) адреса страницы -> байты строки адреса -> количество байт заголовка страницы -> последовательность байт заголовка -> количество байт адреса следующей страницы и так далее.

Добавляем в меню «Избранное» новый пункт:

```
void MainWindow::addToMenuFavorite(QString& key)
{
    QAction *act = favList->addAction(key);
    connect(act, SIGNAL(triggered()), this, SLOT(openFavoritePage()));
}
```

Каждый сигнал пункта меню подключен к одному и тому же слоту. Чтобы различать, от какого пункта поступает сигнал, воспользуемся методом **QObject::sender()**:

```
void MainWindow::openFavoritePage()
{
    QAction *act = static_cast<QAction*>(sender());
    QString s = listFav[act->text()];
    webView->setUrl(QUrl(s)); // Получим значение контейнера по ключу
}
```

Вернемся к компоновке. Напишем код, создающий главное меню:

```
void MainWindow::addMainMenu()
{
    QMenu *mainMenu = new QMenu(this);
    QAction *act = mainMenu->addAction(tr("Save Page as"));
    connect(act, SIGNAL(triggered()), this, SLOT(savePageAs()));
    act = mainMenu->addAction(tr("Quit"));
```

```
    connect(act, SIGNAL(triggered()), this, SLOT(close()));
    menuButton->setMenu(mainMenu);
}
```

Главное меню будет содержать два пункта: сохранение страницы и выход.

```
void MainWindow::savePageAs()
{
    QString filename = QFileDialog::getSaveFileName(this, tr("Save this page"),
QDir::home().path(),
                                                    tr("Hyber text archive") +
"(*.mht);;" + tr("All files") + "(*.*)");
    if (filename.length() == 0) return;
    webView->page()->save(filename);
}
```

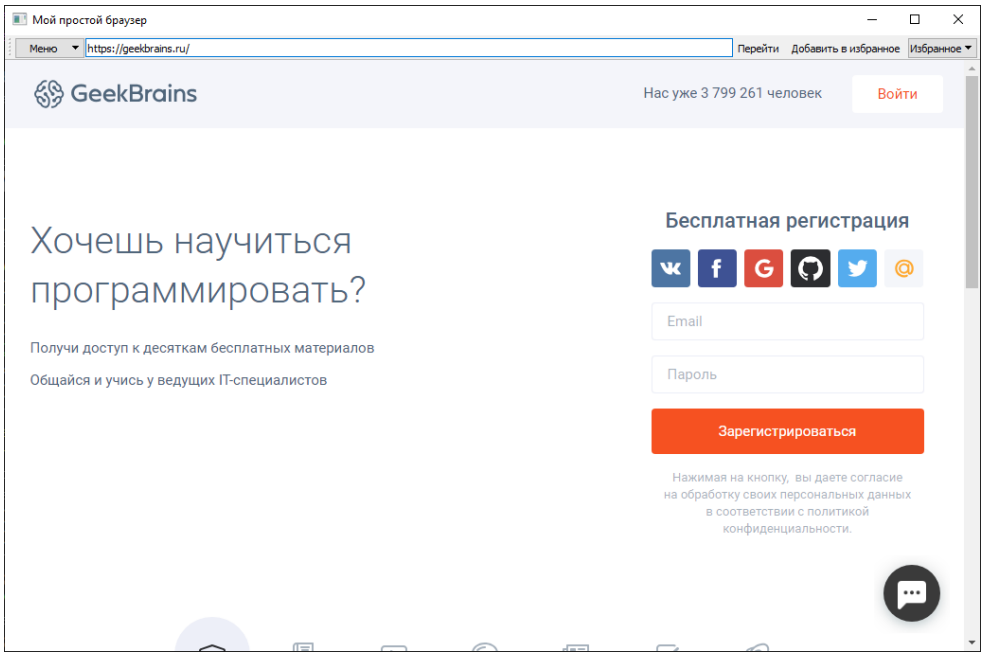
Осталось добавить метод загрузки избранных страниц:

```
void MainWindow::loadFavorites()
{
   QFile file("./favlist.txt");
   if (file.open(QIODevice::ReadOnly))
   {
       for(;!file.atEnd();)
       {
           int length = 0;
           file.read((char*)&length, sizeof length);
           QByteArray ar;
           ar.resize(length);
           file.read(ar.data(), length);
           QString url(ar);

           file.read((char*)&length, sizeof length);
           ar.resize(length);
           file.read(ar.data(), length);
           QString key = QString::number(listFav.count() + 1) + ". " +
QString(ar);                        // Создаем уникальное имя заголовка
           listFav[key] = url;      // Добавляем в контейнер
           addToMenuFavorite(key); // Добавляем в меню “Избранное”
       }
       file.close();
   }
}
```

Запустим утилиту **lupdate** и отредактируем локализацию русского языка, добавив скомпилированный файл локализации в ресурсы. Загрузим локализацию при запуске программы **main.cpp**:

```
#include "mainwindow.h"
```

```cpp
#include <QApplication>
#include <QTranslator>

int main(int argc, char *argv[])
{
    QApplication a(argc, argv);
    QTranslator translatot;
    translatot.load(":/browser_" + QLocale::system().name());
    a.installTranslator(&translatot);
    MainWindow w;
    w.setWindowTitle(QApplication::tr("My simple browser"));
    w.setWindowState(Qt::WindowState::WindowMaximized);
    w.show();

    return a.exec();
}
```



Практическое задание

1. В проект "Текстовый редактор" (можно взять из предыдущих ДЗ) добавить:
    a. возможность копировать формат фрагмента текста и применять к другому фрагменту.
    b. возможность выравнивания текста по правому и левому краю, а также по центру.
    c. возможность выбора шрифта
2. Используя графическое представление, создать программу-окно, на которое можно добавлять графические элементы щелчком левой кнопки мыши на пустой области окна. Добавляемыми элементами могут быть прямоугольник, эллипс, звезда, они должны добавляться по очереди, т.е. первый щелчок

мыши - прямоугольник, второй - эллипс, третий - звезда, четвертый - снова прямоугольник и т.д. Цвет этих фигур должен быть случайным. При щелчке правой кнопки мыши по элементу он должен удалиться. Должна быть возможность перемещать элементы, с помощью зажатой левой кнопкой мыши.

3. * Для предыдущей задачи добавить:
    a. Возможность приближать и удалять изображение путем прокручивания колеса мыши и нажатия клавиш + или - соответственно.
    b. Возможность вращать фигуру вокруг своего центра зажатым колесом мыши.

Дополнительные материалы

1. Список основных тегов HTML. http://www.astro.spbu.ru/staff/afk/Teaching/Help/Tegs.htm
2. Основные методы QCharTextFormat. https://doc.qt.io/qt-5/qtextcharformat.html
3. Классы модуля Qt WebEngine. https://doc.qt.io/qt-5/qtwebenginecore-module.html

Используемая литература

Для подготовки данного методического пособия были использованы следующие ресурсы:

1. Официальная документация QCharTextFormat. https://doc.qt.io/qt-5/qtextcharformat.html
2. Официальная документация QPainter. https://doc.qt.io/qt-5/qpainter.html
3. Классы модуля Qt WebEngine. https://doc.qt.io/qt-5/qtwebenginecore-module.html