Практическое задание №2.

Дополнение демонстрационного приложения для работы с библиотекой подсветки QSourceHighlighter в среде QtCreator.

Цель работы: научиться использовать стандартные диалоговые окна библиотеки Qt и создавать пользовательские диалоги на основе QDialog, а также освоить работу с форматом JSON.

Задание:

Клонировать open-source проект https://github.com/Waqar144/QSourceHighlite и добавить новый функционал:

- Пункт главного меню «Файл», содержащий следующие действия:
  - "Сохранить как..." → TXT или JSON
  - "Открыть как…" → текст или файл JSON
  - "Выход" с предложением сохранить изменения перед выходом.
- Пункт главного меню «Правка», содержащий следующие действия:
  - "Копировать"
  - "Вставить"
  - "Вырезать"
  - "Очистить"
  - "Отменить"
  - "Повторить"
  - "Поиск" → открытие окна поиска.
- Диалоговое окно «Поиск», в котором содержится поле ввода строки поиска и кнопки «Найти далее», «Заменить», «Заменить все».
- Сохранение документа в формате JSON должно позволять сохранять установленный пользователем язык программирования и текст.
- Загрузка файла формата JSON должна позволять восстанавливать текст и язык программирования в интерфейсе приложения.
- Строка состояния, отображающая временные сообщения о выполненных действиях программы.

Порядок выполнения:

1) Соберите проект в Qt Creator и убедитесь, что он запускается. Функционал смены темы оформления данная работа не затрагивает.
2) Исправьте метод смены языка в выпадающем списке, оставив в нем только два действия:

```
void MainWindow::languageChanged(const QString &lang) {
    highlighter->setCurrentLanguage(_langStringToEnum.value(lang)); //выбор языка
    highlighter->rehighlight(); //смена подсветки
}
```

Рисунок 16. Установка подсветки текста.

3) Добавьте пункты в главное меню по заданию.
4) Ряд действий из меню «Правка» содержится в контекстном меню виджета plainTextEdit. Свяжите имеющиеся действия с пунктами главного меню, используя режим редактирования сигналов и слотов:

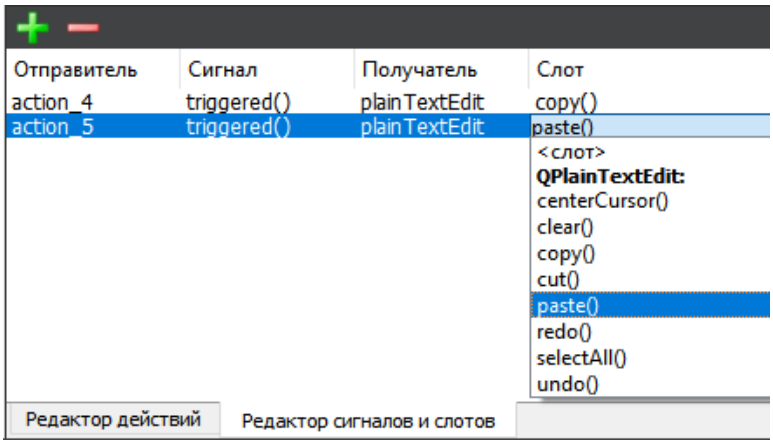

Рисунок 17. Редактирование сигналов и слотов.

5) Реализуйте сохранение обычного текстового файла, выводя в панель состояния временные сообщения о результатах действия. Например, при успешном сохранении файла нужно вывести информацию о том, какой файл был сохранен.
6) Реализуйте сохранение файла в формате JSON, воспользовавшись классами QJsonObject и QJsonDocument. Сохранять следует текущий выбранный язык из выпадающего списка и текст из поля ввода:

```
void MainWindow::on_actionJSON_triggered()
{
    QString fileName = QFileDialog::getSaveFileName(this,
                                                     "Сохранить JSON файл",
                                                     "",
                                                     "JSON файлы (*.json)");

        if (fileName.isEmpty()) return;

        QFile file(fileName);
        if (!file.open(QIODevice::WriteOnly | QIODevice::Text)) {
            ui->statusbar->showMessage("Ошибка: не удалось открыть файл для записи", 3000);
            return;
        }

        QJsonObject root;
        root["language"] = ui->langComboBox->currentText();
        root["text"] = ui->plainTextEdit->toPlainText();

        QJsonDocument doc(root);
        file.write(doc.toJson());
        file.close();
        ui->statusbar->showMessage("Сохранен файл "+fileName, 3000);

}
```

Рисунок 18. Слот сохранения файла в формате JSON.

В панель состояния также необходимо выводить временные сообщения о результатах данного действия.

7) Реализуйте открытие текстового файла, выводя временные сообщения в строку состояния о том, какой файл был открыт.
8) Реализуйте открытие файла формата JSON, используя классы QJsonDocument и QJsonParseError:

```
QJsonParseError error;
QJsonDocument doc = QJsonDocument::fromJson(file.readAll(), &error);
file.close();

if (error.error != QJsonParseError::NoError) {
    ui->statusbar->showMessage("Ошибка парсинга JSON", 3000);
    return;
}
```

Рисунок 19. Фрагмент кода для чтения и парсинга данных JSON.

Для извлечения данных используйте классы QJsonObject и QJsonValue, получая корневой JSON объект и извлекая значение по ключу для установки текста и языка в редактор:

```
QJsonObject obj = doc.object();
QString lang = obj["language"].toString();
QString text = obj["text"].toString();

// Установка текста
ui->plainTextEdit->setPlainText(text);

// Установка языка
if (!lang.isEmpty()) {
    ui->langComboBox->setCurrentText(lang);
}
```

Рисунок 19. Фрагмент кода для извлечения данных JSON.

9) Реализуйте функционал меню для выхода из приложения с проверкой несохраненных изменений, предлагая пользователю сохранить файл при необходимости. Используйте класс QMessageBox для вывода диалогового окна и QTextDocument::isModified() для проверки на изменения.

10) Создайте диалог поиска и замены, следуя рекомендациям:

- Добавьте новый класс C++ , например, SearchDialog, который будет наследоваться от QDialog.

- В заголовочном файле определите необходимые элементы управления, например, QLineEdit для ввода строк поиска и замены, QPushButton для кнопок "Найти далее", "Заменить" и «Заменить все».

- Данный диалог должен отвечать только за сбор данных для поиска и замены, а действия над текстом реализованы в MainWindow. Соответственно в заголовочный файл диалога добавьте сигналы для передачи данных в главное окно приложения:

```
signals:
    void findNext(const QString &text);//поиск
    void replaceText(const QString &find, const QString &replace);//замена
    void replaceAll(const QString &find, const QString &replace);//заменить все
```

Рисунок 20. Сигналы для связи между диалоговым окном поиска и замены и редактором текста.

- Реализуйте слоты, которые будут вызываться при нажатии кнопок в диалоговом окне поиска: для кнопки «Найти» обработчик нажатия будет отправлять сигнал findNext главному окну со строкой для поиска в редакторе; для кнопок «Заменить» и «Заменить все» - отправка сигнала replaceText и replaceAll со строками для поиска и замены:

```
void SearchDialog::on_findButton_clicked() {
    emit findNext(findLineEdit->text());
}

void SearchDialog::on_replaceButton_clicked() {
    emit replaceText(findLineEdit->text(), replaceLineEdit->text());
}

void SearchDialog::on_replaceAllButton_clicked() {
    emit replaceAll(findLineEdit->text(), replaceLineEdit->text());
}
```

Рисунок 21. Вызов сигналов при работе с диалоговым окном поиска

Также объявите данные слоты в заголовочном файле диалога в секции private slots и свяжите их с соответствующими кнопками функцией connect() в конструкторе диалога.

- Добавьте соответствующие слоты в MainWindow, объявив их в заголовочном файле. Слот для поиска должен искать следующее совпадение с текущей позиции. Если совпадение не найдено – продолжить поиск с начала документа. Если не найдено совпадений во всем документе – выводить об этом сообщение. Пример реализации слота поиска:

```
void MainWindow::onFindText(const QString &text) {
     // Получаем курсор из QPlainTextEdit (текущую позицию ввода)
    QTextCursor cursor = ui->plainTextEdit->textCursor();

    // Получаем весь текст из редактора как QString для поиска
    QString content = ui->plainTextEdit->toPlainText();

    // Ищем текст, начиная с текущей позиции курсора
    int pos = content.indexOf(text, cursor.position(), Qt::CaseSensitive);

        if (pos >= 0) {
            // Нашли совпадение → устанавливаем курсор на него
            cursor.setPosition(pos);
            cursor.movePosition(QTextCursor::Right, QTextCursor::KeepAnchor, text.length());
            ui->plainTextEdit->setTextCursor(cursor);// выделяем найденный фрагмент
            ui->statusbar->showMessage(tr("Найдено: ") + text, 3000);
        } else {
              // Совпадений нет от текущей позиции до конца, начинаем поиск с самого начала
            pos = content.indexOf(text, 0, Qt::CaseSensitive);
            if (pos >= 0) {
                // Нашли совпадение при повторном поиске с начала
                cursor.setPosition(pos);
                cursor.movePosition(QTextCursor::Right, QTextCursor::KeepAnchor, text.length());
                ui->plainTextEdit->setTextCursor(cursor);
                ui->statusbar->showMessage("Начали поиск с начала. Найдено: " + text, 3000);
            } else {
                // Ничего не найдено даже после повторного поиска
                QMessageBox::information(this, "Поиск", "Совпадений не найдено");
                ui->statusbar->showMessage(tr("Совпадений не найдено"), 3000);
            }
        }
}
```

Рисунок 22. Слот поиска совпадения

Слот для однократной замены должен заменять одно вхождение текста, заданного пользователем. Сначала нужно проверить, выделено ли нужное слово и выполнить замену, если совпадение уже найдено. Если слово не выделено, то должен производиться поиск следующего вхождения искомой строки с текущей позиции курсора. После замены положение курсора должно обновляться, чтобы пользователь видел, где произошли изменения. Затем нужно обновить подсветку. Пример реализации:

```
void MainWindow::onReplaceText(const QString &find, const QString &replace) {
    // Получаем указатель на документ
    QTextDocument *doc = ui->plainTextEdit->document();

    // Получаем текущий курсор
    QTextCursor cursor = ui->plainTextEdit->textCursor();

    // Проверяем, есть ли выделенный текст
    QString selected = cursor.selectedText();
```

```
    // Если есть выделение, и оно совпадает с find - заменяем его
    if (!selected.isEmpty() && selected == find) {
            cursor.insertText(replace);// Заменяем выделенный текст
            ui->plainTextEdit->setTextCursor(cursor);// Перемещаем курсор
            return;
        }

        int pos = cursor.position();
      // Ищем следующее вхождение с учетом регистра
        QTextCursor found = doc->find(find, pos, QTextDocument::FindCaseSensitively);

        if (!found.isNull()) {
            // Совпадение найдено - сохраняем начальную и конечную позиции курсора
            int start = found.selectionStart();
            int end = found.selectionEnd();

            // Замена найденного текста
            found.insertText(replace);

            // Создаем новый курсор, чтобы переместить выделение на место замены
            QTextCursor newCursor = ui->plainTextEdit->textCursor();

            // Устанавливаем курсор на начало замены
            newCursor.setPosition(start);
            // Выделяем новую строку длиной replace.length()
            newCursor.movePosition(QTextCursor::Right, QTextCursor::KeepAnchor, replace.length());
            // Перемещаем курсор в UI
            ui->plainTextEdit->setTextCursor(newCursor);

        } else {
            QMessageBox::information(this, "Поиск","Совпадений не найдено");
        }

    highlighter->rehighlight(); // обновляем подсветку
}
```

Рисунок 23. Слот замены

Слот для замены всех вхождений выполняет замену заданной строки и обновляет содержимое редактора, а также выводит сообщение в строку состояния:

```
void MainWindow::onReplaceAll(const QString &find, const QString &replace) {
    QString content = ui->plainTextEdit->toPlainText();
    // Заменяем все вхождения find на replace с учётом регистра
    QString newContent = content.replace(find, replace, Qt::CaseSensitive);

    // Устанавливаем обновлённый текст обратно в редактор
    ui->plainTextEdit->setPlainText(newContent);
    highlighter->rehighlight();

    QString message = (QString("Все вхождения '%1' заменены на '%2'").arg(find).arg(replace));
    ui->statusbar->showMessage(message, 3000);
}
```

Рисунок 24. Слот замены всех совпадений

- Реализуйте обработчик вызова диалога поиска из главного меню программы. В данном обработчике необходимо создать экземпляр реализованного диалогового окна и добавить связи сигналов диалога со слотами главного окна:

```
void MainWindow::on_action_10_triggered()
{
    SearchDialog dialog(this);
    connect(&dialog, &SearchDialog::findNext, this, &MainWindow::onFindText);
    connect(&dialog, &SearchDialog::replaceText, this, &MainWindow::onReplaceText);
    connect(&dialog, &SearchDialog::replaceAll, this, &MainWindow::onReplaceAll);
    dialog.exec(); // показываем модальное окно
}
```

Рисунок 24. Обработчик вызова окна поиска и замены

11) Протестируйте работу полученного приложения, проверив следующие случаи:

- Сохраните файл в формате .txt и .json и убедитесь, что statusbar выдает сообщение о пути сохранения.
- Проверьте, что в .json файле сохраняются выбранный язык программирования и сам текст., в в .txt файле сохраняется только текст без метаданных.
- Откройте .txt и .json файлы в редакторе и убедитесь, что при открытии .json язык программирования и соответствующая подсветка восстанавливается, а текст корректно загружается в редактор.
- Если вы открываете поврежденный файла JSON, то должно появиться сообщение об ошибке.
- Поменяйте язык в выпадающем списке и убедитесь, что ключевые слова, строки, комментарии и числовые литералы подсвечиваются правильно.
- Откройте любой текст в редакторе и нажмите "Поиск" в меню "Правка", введите строку для поиска и проверьте работает ли кнопка "Найти далее", подсвечивается ли найденный текст, возможна ли замена одного фрагмента и всех вхождений
- Проверьте, что при достижении конца документа поиск продолжается с начала.
- Внесите изменения в текст, но не сохраняйте его, нажмите "Выход" . Должно появиться окно с вопросом о сохранении документа. При выборе "Да" → открывается диалог

сохранения, "Нет" → приложение закрывается без сохранения, "Отмена" → выход отменяется, возврат к редактору.

- Проверьте, что все остальные пункты меню работают как ожидается: кнопки "Копировать", "Вставить", "Вырезать", "Очистить" работают как в контекстном меню.
- Проверьте работу statusbar: временные сообщения должны появляться и исчезать через 3 секунды.

Дополнительное задание: Добавить иконки к пунктам главного меню, а также реализовать быструю панель инструментов (toolbar), содержащую часто используемые действия: открытие файла, сохранение, поиск.

Контрольные вопросы:

1. Как работает механизм подсветки синтаксиса в QSyntaxHighlighter?
2. Что делает метод rehighlight() и почему он важен при изменении языка или текста?
3. Какие сигналы и слоты используются в диалоге поиска и замены?
4. Что хранится в JSON-файле при сохранении документа?
5. Как восстанавливается язык программирования при открытии JSON-файла?
6. Как работает связь между ComboBox и подсветкой кода?
7. Какие классы Qt используются для создания пользовательского диалога поиска и замены?
8. Как применяется QTextCursor в данном проекте?
9. Как проверить, были ли сделаны изменения в документе перед выходом?
10. Какие классы и методы Qt используются для работы с JSON?