Правительство Российской Федерации

Федеральное государственное автономное образовательное учреждение высшего образования «Национальный исследовательский университет «Высшая школа экономики»

Кафедра «Компьютерная безопасность»

ОТЧЁТ
К ЛАБОРАТОРНОЙ РАБОТЕ №12
по дисциплине
«Языки программирования»

Работу выполнил студент группы СКБ-193 _____ Подпись, дата — Д.И. Лежнин

Работу проверил _____ Подпись, дата — С.А. Булгаков

Москва 2020

Постановка задачи

Разработать графическое приложение с использованием библиотеки Qt–текстовый редактор:

1)с подсветкой текущей строки;
2)с нумерацией строк;
3)с подсветкой синтаксиса (переключение): Си89, Си++98/03

Заголовок окна должен содержать имя редактируемого файла (ограниченное 32 символами + троеточие) и признак того что файл был изменен (звездочка в начале имени + курсив) с момента последнего сохранения.

Окно содержит главное меню (и наследуется от QMainWindow) состоящее из пунктов:

1) Файл [кнопки]
    a) Новый
    b) Открыть
    c) Сохранить
    d) Сохранить как
    e) Выход
2) Правка [кнопки]
    a) Отменить
    b) Повторить
    c) Копировать
    d) Вырезать
    e) Вставить
    f) Найти
    g) Найти и заменить
    h) Выделить все
3) Формат
    a) Перенос по словам [галочка]
    b) Выбор шрифта -открывается модальный диалог выбора шрифта
4) Вид [галочки, где не указано иное]
    a) Выбор цвета фона [кнопка] -открывает модальное окно выбора цвета
    b) Выбор цвета текущей строки [кнопка] -открывает модальное окно выбора цвета
    c) Вкл/Выкл отображения нумерации строк
    d) Вкл/Выкл отображения панели инструментов
    e) Вкл/Выкл отображения строки состояния
    f) Вкл/Выкл подсветки синтаксиса
    g) Выбор синтаксиса (для подсветки)[дочернее меню] -доступно всегда, один синтаксис в дочернем меню выбран всегда
    h) Выбор/Редактирование стиля подсветки [дочернее меню] -для текущего синтаксиса, по умолчанию выбран Default
    i) Изменить [кнопка] -измененный стиль сохраняется в файл, имя файла становится именем стиля, стиль становится активным
    ii) Загрузка стиля из файла [кнопка] -имя файла становится именем стиля, стиль становится активным

iii) Обязательная кнопка Default

iv) доступные стили[перечисляются все стили которые были обнаружены]

5) Справка

а) О программе -открывает модальное окно содержащее фото и имя автора, дату сборки, версию Qt с которой собиралось, версию Qt с которой запущено, кнопку закрывающую окно

Ниже главного меню располагается панель инструментов(отображение которой контролируется в меню Вид)с кнопками (с картинками, текстовое описание во всплывающей подсказке):

1) Новый документ
2) Открыть
3) Сохранить
4) Отменить
5) Повторить
6) Копировать
7) Вырезать
8) Вставить
9) Найти / Найти и заменить (как выпадающая кнопка) -открывающая (немодальное)диалоговое окно

В центральной части окна располагается область для редактирования текста. При нажатии левой кнопки курсор вставляется в позицию. При двойном нажатии левой кнопки выделяется слово под курсором. При нажатии правой кнопки(далее -если нет=*, есть=** выделения)курсор вставляется в позицию и выдается контекстное меню (кнопки могут быть неактивны): отменить, повторить, выделить*, выделить строку*, копировать**, вырезать**, вставить (** или если есть текст в буфере обмена), удалить**, выделить все.
Нижнюю часть окна занимает строка состояния. Информация разделена на три столбца: текущая позиция курсора (строка:столбец); время (и дата если другие сутки) последней операции (сохранения/изменения); количество строк, слов, символов, размер в килобайтах.

Дополнительный балл –подсветка синтаксиса Си++11/14 (Си++17, Си++20)

Дополнительный балл –сохранение настроек приложения в ini-файл.

1. Алгоритм решения задачи

Данная программа состоит из файлов, в которых определены и реализованы пользовательские классы для решения поставленных задач: **AboutModal, EditArea, LineNumberArea, FindAndReplaceModal, FindModal, Highlighter, StyleModal, Textedit**. Также, программа содержит главный файл (main.cpp), в котором подключается класс **Textedit** и запускается событийный цикл. Кроме того, в программе присутствует файл проекта (lb12.pro), который содержит информацию, необходимую qmake для сборки проекта и файл коллекции ресурсов (images.qrc). Также в программе содержатся изображения и файлы стилей в папках images и styles соответственно. Основные связи между классами представлены на UML-диаграмме (рисунке 1).

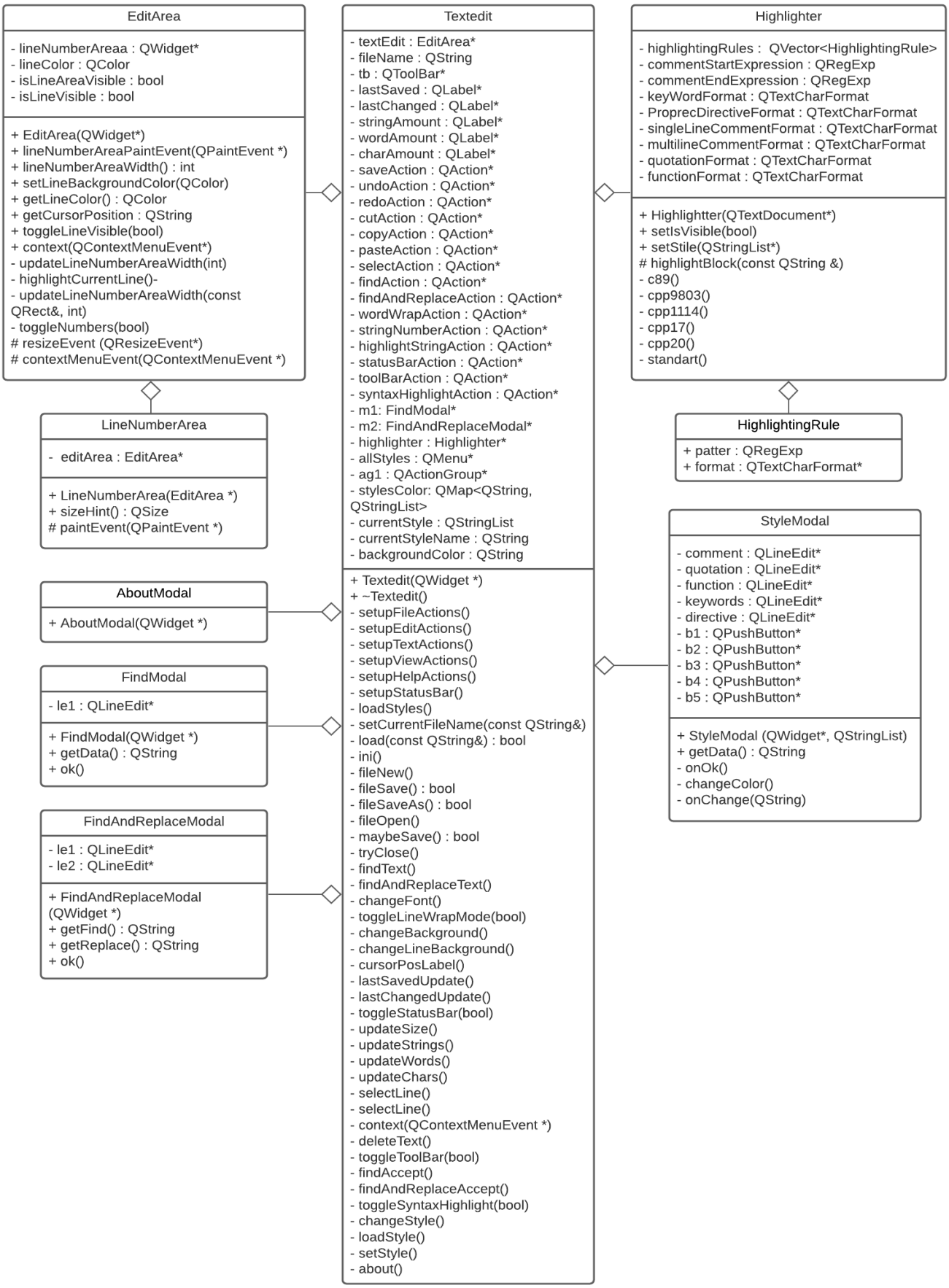

Рисунок 1 – UML-диаграмма классов

2.Выполнение задания

2.1. Файлы textedit.h и textedit.cpp

В данных файлах определён и реализован класс **Textedit,** являющийся главным окном приложения. Содержит в себе функции для настройки главного меню приложения (**setupFileActions(), setupEditActions(), setupTextActions(), setupViewActions(), setupHelpAction()**), в которых создаются доступные в приложении действия – объекты класса **QAction** (**saveAction, undoAction, redoAction, cutAction, copyAction, pasteAction, selectAllAction, findAction, findAndReplaceAction, wordWrapAction, wordWrapAction, stringNumberAction, highlightStringAction, statusBarAction, toolBarAction, syntaxHighlightAction**) – и связываются с функциями, которые их обрабатывают (**fileSave(), fileSaveAs(), fileOpen(), findText(), findAndReplaceText(), ChangeFonT(), toggleLineWrapMode(), changeBackground(), selectWord(), selectLine(), deleteText()**).

Для переключения состояния отображения элементов интерфейса – строки состояния, панели инструментов, подсветки синтаксиса – созданы функции: **toggleStatusBar(), toggleToolBar(), ToggleSyntaxHighlight().**

Данные в строке состояния обновляются с помощью функций: **updateSize(), updateStrings(), updateChars(), updateWords(), cursorPosLabel(), lastSavedUpdate(), lastChangedUpdate()**.

В функции **context(QContextMenuEvent*)** создаётся контекстное меню, вызываемое при клике правой кнопкой мыши. Доступные в нём действия зависят от того, есть выделения текста или нет.

В классе присутствуют функции для загрузки, сохранения и переключения стиля подсветки(**loadStyle(), changeStyle(), setStyle()**). Данные о всех стилях хранятся в **QMap<QString, QStringList> stylesColors**. Текущий стиль и его название хранятся в **QStringList currentStyle** и **QString currentStyleName** соответственно.

Также при закрытие приложения некоторые его настройки сохранятся в ini-файл в деструкторе при помощи объекта класса **QSettings**, а загружаются при открытии приложения в функции **ini().**

Кроме того, в классе присутствуют вспомогательные функции: **tryClose(), maybeSave()**

2.2. Файлы editarea.h и editarea.cpp

В данных файлах определён и реализован класс **EditArea,** являющийся полем для редактирования текста и наследующийся от **QPlainTextEdit**. В функции **lineNumberAreaPaintEvent()** происходит рисования прямоугольника, являющегося нумерацией строк, ширина которого вычисляется в функции **lineNumberAreaWidth(),** а обнавляется в **updateLineNumberAreaWidth()**. Отключение нумерации строк происходит в функции **tooggleNumbers()**.

В функции **highlightCurrentLine()** происходит подсветка текущей строки, её цвет меняется в функции **setLineBackground()**, а её отключение происходит в функции **toggleLineVisible().**

Также в классе присутствуют вспомогательные функции для получения текущей позиции курсора и цвета подсветки текущей строки.

Кроме того, в данных файлах присутствует класс **LineNumberArea**, который представляет область нумерции строк, с функцией рисования **paintEvent(QPaintEvent *)**

2.3. Файлы highlighter.h и highlighter.cpp

В данных файлах определён и реализован класс **Highlighter,** который представляет функционал для подсветки синтаксиса. Правила для подсветки строк, однострочных комментариев, многострочных комментариев, ключевых слов, директив препроцессора и функций определены в объектах класса **QTextCharFormat (singleLineCommentFormat, multiLineCommentFormat, keywordFormat, PreprocDirectiveFormat, quotationFormat, functionFormat).**

Сама подсветка происходит в функции **highlightBlock().** Подсветка реализована для синтаксисов: c89, си++98/03, си++11/14, си++17, си++20 и переключается в функциях **c89(), cpp9803(), cpp1114(), cpp17(), cpp20() соответственно.** Отключение подсветки происходит в функции **setIsVisible().** Изменение цветов для подсветки происходит в функции **setStyle(QStringList).** Также в классе присутствует функция **standart()**, в которой задаются стандартные для всех синтаксисов правила.

2.4. Файлы findmodal.h, findmodal.cpp

В данных файлах определен и реализован класс **FindModal**, являющийся окном для поиска слов в тексте. Введённый текст хранится в объекте класса **QLineEdit le1.** Также в классе присутствуют Кнопки для поиска и отмены – объекты класса **QPushButton**. При нажатии на кнопку поиска создаётся сигнал **ok().** Также присутствует функцию **getData()** для получения значения из окна.

2.5. Файлы findandreplacemodal.h, findamdreplacemodal.cpp

В данных файлах определен и реализован класс **FindAndReplaceModal**, являющийся окном для поиска и замены слов в тексте. Введённый текст хранится в объекте класса **QLineEdit le1,** а текст для замены в **QLineEdit le2.** Также в классе присутствуют Кнопки для поиска и отмены – объекты класса **QPushButton**. При нажатии на кнопку поиска создаётся сигнал **ok().** Также присутствуют функцию **getFind() и getReplace()** для получения данных из окна.

2.6. Файлы stylemodal.h и stylemodal.cpp

В данных файлах определён и реализован класс **StyleModal,** который является модальным окном для изменения стиля подстветки. Данных о цветах для подсветки строк, однострочных комментариев, многострочных комментариев, ключевых слов, директив препроцессора и функций хранятся в объектах класса QLineEdit (**quotation, comment, keyword, directive, и function**). Также для каждого цвета присутствует кнопка, при нажатии на которую открывается диалоговое окно для выбора цвета, цвет фона кнопки соответствует выбранному цвету(**QPushButton* b1, b2, b3, b4, b5**). Для получения данных из окна используется функция **getData().**

2.7. Файлы aboutmodal.h, aboutmodal.cpp

В данных файлах определён и реализован класс AboutModal, являющийся модальным окном, в котором отображается основная информация о приложения: имя автора, фото, версия Qt, с которой собиралось, версия Qt, с которой запускалось и дату сборки.

2.8. Файл main.cpp

В данном файле подключается заголовочный файл класса **Textedit** и создаётся его экземпляр для функционирования всего приложения. Также создаётся экземпляр класса **QApplication** и запускается событийный цикл.

2.9. Файл lab12.pro

Данный файл содержит информацию о конфигурации всего приложения.

2.10. Файл images.qrc

Данный файл содержит в себе информацию о подключаемых в приложении ресурсахю.

3. Процедура получения исполняемых файлов

Программа снабжена файлом lab12.pro, который обрабатывается с помощью утилиты qmake. При этом генерируется makefile. в котором заданы правила сборки программы.

4. Тестирование программы

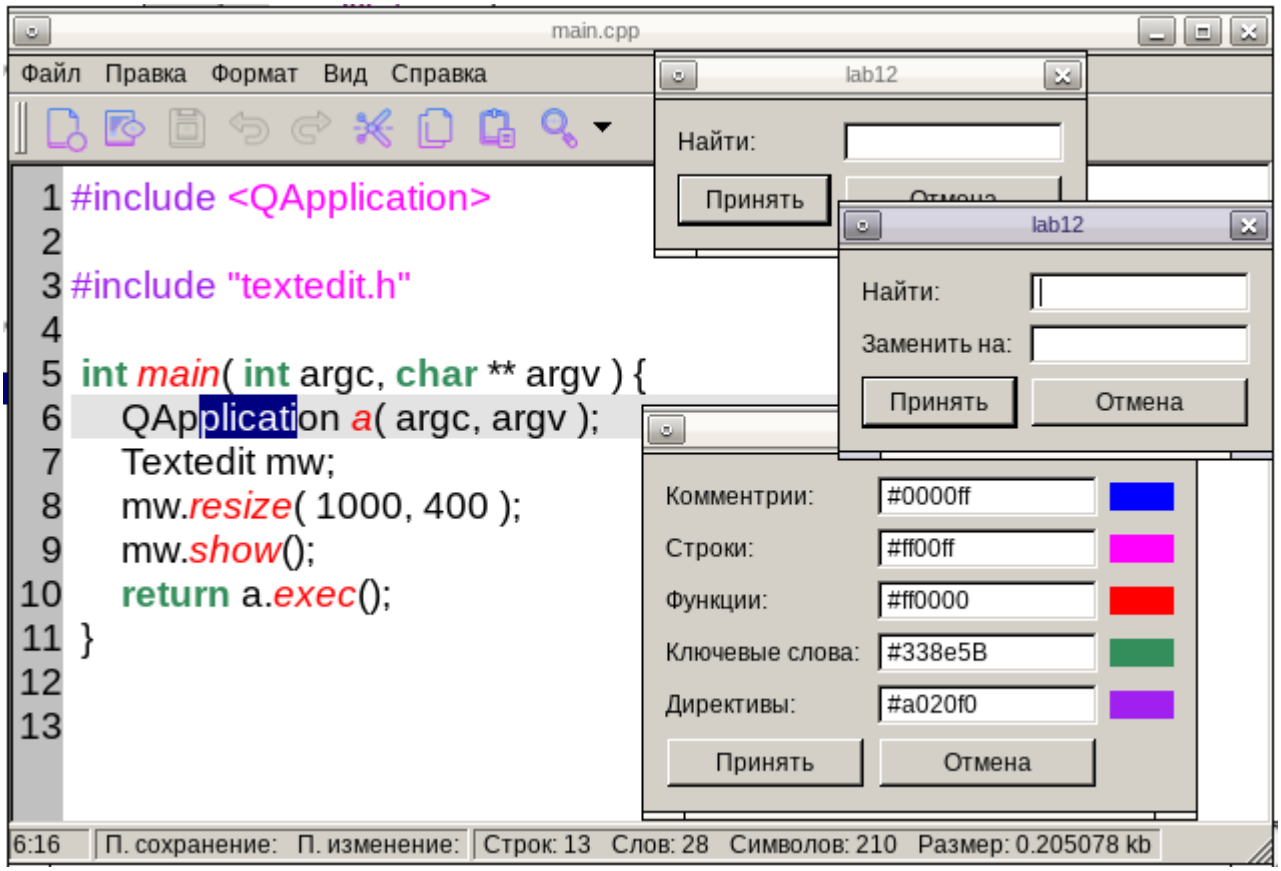

Рисунок 2 – Тестирование программы

Приложение А

А1. Исходный код файла **main.cpp**

```
#include <QApplication>

#include "textedit.h"

 int main( int argc, char ** argv ) {
     QApplication a( argc, argv );
     Textedit mw;
     mw.resize( 1000, 400 );
     mw.show();
     return a.exec();
 }
```

А2. Исходный код файла **textedit.h**

```
#ifndef TEXTEDIT_H
#define TEXTEDIT_H

#include <QMainWindow>
//#include <QPlainTextEdit>
#include <QMenu>
#include <QMenuBar>
#include <QMessageBox>
#include <QTextDocumentWriter>
#include <QFileDialog>
#include <QFontDialog>
#include <QColorDialog>
#include <QDebug>
#include <QStatusBar>
#include <QHBoxLayout>
#include <QSplitter>
#include <QTime>
#include <QToolBar>
#include <QToolButton>
#include <QDir>
#include <iostream>
#include <QDateTime>
#include <QSettings>
#include "highlighter.h"
#include "editarea.h"
#include "findmodal.h"
#include "findandreplacemodal.h"
#include "stylemodal.h"
#include "aboutmodal.h"

class Textedit : public QMainWindow
{
    Q_OBJECT
public:
    Textedit(QWidget *parent = 0);
    ~Textedit();
private:
```

```
    void setupFileActions();
    void setupEditActions();
    void setupTextActions();
    void setupViewActions();
    void setupHelpAction();
    void setupStatusBar();
    void loadStyles();
    void setCurrentFileName(const QString &);
    bool load(const QString& f);
    void ini();
    EditArea* textEdit;
    QString fileName;
    QToolBar *tb;
    QLabel* cursorPosition,
        *lastSaved,
        *lastChanged,
        *stringAmount,
        *wordAmount,
        *charAmount,
        *documentSize;

    QAction *saveAction,
    *undoAction,
    *redoAction,
    *cutAction,
    *copyAction,
    *pasteAction,
    *selectAllAction,
    *findAction,
    *findAndReplaceAction,
    *wordWrapAction,
    *stringNumberAction,
    *highlightStringAction,
    *statusBarAction,
    *toolBarAction,
    *syntaxHighlightAction;

    FindModal *m1;
    FindAndReplaceModal *m2;

    Highlighter *highlighter;
/*''*/
    QMenu *allStyles;
    QActionGroup  *ag1;

    QMap<QString, QStringList> stylesColors;
    QStringList currentStyle;
    QString currentStyleName;
    QString backgroundColor;
protected:
    void closeEvent(QCloseEvent * event);
private slots:
    void fileNew();
    bool fileSave();
    bool fileSaveAs();
    void fileOpen();
```

```
    bool maybeSave();
    void tryClose();
    void findText();
    void findAndReplaceText();
    void /*c*/ChangeFonT/*t*/();
    void toggleLineWrapMode(bool f);
    void changeBackground();
    void changeLineBackground();
    void cursorPosLabel();
    void lastSavedUpdate();
    void lastChangedUpdate();
    void toggleStatusBar(bool f);
    void updateSize();
    void updateStrings();
    void updateChars();
    void updateWords();
    void selectWord();
    void selectLine();
    void context(QContextMenuEvent *);
    void deleteText();
    void toggleToolBar(bool f);
    void findAccept();
    void findAndReplaceAccept();
    void toggleSyntaxHighlight(bool f);
    void changeStyle();
    void loadStyle();
    void setStyle();
    void about();
signals:

public slots:

};

#endif // TEXTEDIT_H
```

А3. Исходный код файла **textedit.cpp**

```
#include "textedit.h"

Textedit::Textedit(QWidget *parent) :
    QMainWindow(parent) {
    m1 = 0;
    m2 = 0;

    setToolButtonStyle(Qt::ToolButtonFollowStyle);
    textEdit = new EditArea(this);
    tb = new QToolBar(this);


    highlighter = new Highlighter(textEdit->document());

    addToolBar(tb);
    setupFileActions();
    setupEditActions();
    setupTextActions();
```

```
    setupViewActions();
    setupStatusBar();
    setupHelpAction();
    loadStyles();
    setCentralWidget(textEdit);
    textEdit->setFocus();


    QMenu *tbm = new QMenu;
    tbm->addAction(findAction);
    tbm->addAction(findAndReplaceAction);
    QToolButton *qtb = new QToolButton();
    qtb->setDefaultAction(findAction);
    qtb->setMenu(tbm);
    qtb->setPopupMode(QToolButton::MenuButtonPopup);
    connect(qtb, SIGNAL(triggered(QAction*)), qtb,
SLOT(setDefaultAction(QAction*)));
    tb->addWidget(qtb);
    ini();

    connect(textEdit->document(), SIGNAL(modificationChanged(bool)),
            this, SLOT(setWindowModified(bool)));
    connect(textEdit->document(), SIGNAL(modificationChanged(bool)),
            saveAction, SLOT(setEnabled(bool)));
    connect(textEdit->document(), SIGNAL(undoAvailable(bool)),
            undoAction, SLOT(setEnabled(bool)));
    connect(textEdit->document(), SIGNAL(redoAvailable(bool)),
            redoAction, SLOT(setEnabled(bool)));
    connect(textEdit, SIGNAL(cursorPositionChanged()), this,
SLOT(cursorPosLabel()));
    connect(textEdit, SIGNAL(textChanged()), this,
SLOT(lastChangedUpdate()));
    connect(textEdit, SIGNAL(textChanged()), this, SLOT(updateStrings()));
    connect(textEdit, SIGNAL(textChanged()), this, SLOT(updateWords()));
    connect(textEdit, SIGNAL(textChanged()), this, SLOT(updateChars()));

    connect(textEdit, SIGNAL(context(QContextMenuEvent*)),
                            this, SLOT(context(QContextMenuEvent*)));

    setWindowModified(textEdit->document()->isModified());
    saveAction->setEnabled(textEdit->document()->isModified());
    undoAction->setEnabled(textEdit->document()->isUndoAvailable());
    redoAction->setEnabled(textEdit->document()->isRedoAvailable());

    connect(undoAction, SIGNAL(triggered()), textEdit, SLOT(undo()));
    connect(redoAction, SIGNAL(triggered()), textEdit, SLOT(redo()));


//    setStyleSheet("background-color: black; color: white;");
}

void Textedit::setupFileActions() {
    QMenu* menu = new QMenu(QString::fromUtf8("Файл"), this);
    menuBar()->addMenu(menu);

    QAction* a;
```

```
    a = new QAction(QIcon(":images/new-file.png"),
QString::fromUtf8("Новый"), this);
    a->setPriority(QAction::LowPriority);
    a->setShortcut(QKeySequence::New);
    connect(a, SIGNAL(triggered()), this, SLOT(fileNew()));
    tb->addAction(a);
    menu->addAction(a);

    a = new QAction(QIcon(":images/open-file.png"),
QString::fromUtf8("Открыть"), this);
    a->setShortcut(QKeySequence::Open);
    connect(a, SIGNAL(triggered()), this, SLOT(fileOpen()));
    tb->addAction(a);
    menu->addAction(a);

    saveAction = new QAction(QIcon(":images/save-file.png"),
QString::fromUtf8("Сохранить"), this);
    saveAction->setShortcut(QKeySequence::Save);
    connect(saveAction, SIGNAL(triggered()), this, SLOT(fileSave()));
    saveAction->setEnabled(false);
    tb->addAction(saveAction);
    menu->addAction(saveAction);

    a = new QAction(QString::fromUtf8("Сохранить как"), this);
    a->setPriority(QAction::LowPriority);
    connect(a, SIGNAL(triggered()), this, SLOT(fileSaveAs()));
    menu->addAction(a);

    a = new QAction(QString::fromUtf8("Выход"), this);
    a->setShortcut(Qt::CTRL + Qt::Key_Q);
    connect(a, SIGNAL(triggered()), this, SLOT(tryClose()));
    menu->addAction(a);
}

void Textedit::setupViewActions() {
    QMenu* menu = new QMenu(QString::fromUtf8("Вид"), this);
    menuBar()->addMenu(menu);

    QAction* a;

    a = new QAction(QString::fromUtf8("Цвет фона"), this);
    connect(a, SIGNAL(triggered()), this, SLOT(changeBackground()));
    menu->addAction(a);

    a = new QAction(QString::fromUtf8("Цвет текущей строки"), this);
    connect(a, SIGNAL(triggered()), this, SLOT(changeLineBackground()));
    menu->addAction(a);

    stringNumberAction = new QAction(QString::fromUtf8("Нумерация строк"),
this);
    stringNumberAction ->setCheckable(true);
    stringNumberAction ->setChecked(true);
    connect(stringNumberAction, SIGNAL(toggled(bool)), textEdit,
SLOT(toggleNumbers(bool)));
```

```
    menu->addAction(stringNumberAction);

    highlightStringAction = new QAction(QString::fromUtf8("Подсветка
строки"), this);
    highlightStringAction->setCheckable(true);
    highlightStringAction->setChecked(true);
    connect(highlightStringAction, SIGNAL(toggled(bool)), textEdit,
SLOT(toggleLineVisible(bool)));
    menu->addAction(highlightStringAction);

    statusBarAction = new QAction(QString::fromUtf8("Строка состояния"),
this);
    statusBarAction->setCheckable(true);
    statusBarAction->setChecked(true);
    connect(statusBarAction, SIGNAL(toggled(bool)), this,
SLOT(toggleStatusBar(bool)));
    menu->addAction(statusBarAction);

    toolBarAction = new QAction(QString::fromUtf8("Панель иснтрументов"),
this);
    toolBarAction->setCheckable(true);
    toolBarAction->setChecked(true);
    connect(toolBarAction, SIGNAL(toggled(bool)), this,
SLOT(toggleToolBar(bool)));
    menu->addAction(toolBarAction);

    syntaxHighlightAction = new QAction(QString::fromUtf8("Подсветка
синтаксиса"), this);
    syntaxHighlightAction->setCheckable(true);
    syntaxHighlightAction->setChecked(true);
    connect(syntaxHighlightAction, SIGNAL(toggled(bool)), this,
SLOT(toggleSyntaxHighlight(bool)));
    menu->addAction(syntaxHighlightAction);


    QMenu *syntax = new QMenu(QString::fromUtf8("Переключить синтаксис"),
menu);
    QActionGroup  *ag = new QActionGroup(syntax);

    a = new QAction(QString::fromUtf8("Си89"), this);
    connect(a, SIGNAL(triggered()), highlighter, SLOT(c89()));
    ag->addAction(a);
    a->setCheckable(true);
    a->setChecked(true);

    a = new QAction(QString::fromUtf8("Си++98/03"), this);
    connect(a, SIGNAL(triggered()), highlighter, SLOT(cpp9803()));
    ag->addAction(a);
    a->setCheckable(true);

    a = new QAction(QString::fromUtf8("Си++11/14"), this);
    connect(a, SIGNAL(triggered()), highlighter, SLOT(cpp1114()));
    ag->addAction(a);
    a->setCheckable(true);

    a = new QAction(QString::fromUtf8("Си++17"), this);
```

```
    connect(a, SIGNAL(triggered()), highlighter, SLOT(cpp17()));
    ag->addAction(a);
    a->setCheckable(true);

    a = new QAction(QString::fromUtf8("Си++20"), this);
    connect(a, SIGNAL(triggered()), highlighter, SLOT(cpp20()));
    ag->addAction(a);
    a->setCheckable(true);

    ag->setExclusive(true);

    syntax->addActions(ag->actions());
    menu->addMenu(syntax);

    QMenu *styles = new QMenu(QString::fromUtf8("Изменить/выбрать стиль"),
menu);

    a = new QAction(QString::fromUtf8("Изменить"), this);
    connect(a, SIGNAL(triggered()), this, SLOT(changeStyle()));
    styles->addAction(a);

    a = new QAction(QString::fromUtf8("Загрузить"), this);
    connect(a, SIGNAL(triggered()), this, SLOT(loadStyle()));
    styles->addAction(a);

    allStyles = new QMenu(QString::fromUtf8("Доступные стили"), styles);
    ag1 = new QActionGroup(allStyles);

    a = new QAction("Default", this);
    a->setObjectName("Default");
    stylesColors["Default"] = {"#0000ff", "#ff00ff", "#ff0000", "#338e5B",
"#a020f0"};
    currentStyle = {"#0000ff", "#ff00ff", "#ff0000", "#338e5B", "#a020f0"};
    currentStyleName = "Default";
    connect(a, SIGNAL(triggered()), this, SLOT(setStyle()));
    a->setCheckable(true);
    a->setChecked(true);
    ag1->addAction(a);

    ag1->setExclusive(true);
    allStyles->addActions(ag1->actions());
    styles->addMenu(allStyles);
    menu->addMenu(styles);

}

void Textedit::setupStatusBar() {
    cursorPosition = new QLabel(" 1:1 ");
    lastSaved = new QLabel(QString::fromUtf8(" П. сохранение: "));
    lastChanged = new  QLabel(QString::fromUtf8(" П. изменение: "));
    stringAmount = new QLabel(QString::fromUtf8(" Строк: "));
    wordAmount = new QLabel(QString::fromUtf8(" Слов: "));
    charAmount = new QLabel(QString::fromUtf8(" Символов: "));
    documentSize = new QLabel(QString::fromUtf8(" Размер: "));

    statusBar()->addWidget(cursorPosition);
```

```
    QSplitter * s1 = new QSplitter;
    QSplitter * s2 = new QSplitter;
    QSplitter * s3 = new QSplitter;

    QSplitter * s21 = new QSplitter;
    QSplitter * s22 = new QSplitter;
    s21->addWidget(lastSaved);
    s22->addWidget(lastChanged);

    s1->addWidget(cursorPosition);

    s2->addWidget(s21);
    s2->addWidget(s22);

    s3->addWidget(stringAmount);
    s3->addWidget(wordAmount);
    s3->addWidget(charAmount);
    s3->addWidget(documentSize);

    statusBar()->addWidget(s1);
    statusBar()->addWidget(s2);
    statusBar()->addWidget(s3);
}

void Textedit::changeStyle() {
    StyleModal *m = new StyleModal(this, currentStyle);
    if (m->exec() == QDialog::Accepted) {
        QStringList style = m->getData();
        if (currentStyleName == "Default") {
            currentStyleName = "Default" +
QString::number(QDateTime::currentMSecsSinceEpoch());
            QAction *a;
            a = new QAction(currentStyleName, this);
            a->setCheckable(true);
            connect(a, SIGNAL(triggered()), this, SLOT(setStyle()));
            a->setObjectName(currentStyleName);
            ag1->addAction(a);
            allStyles->addActions(ag1->actions());
            a->setChecked(true);
    //        qDebug() << colors;
        }
        QString styleString = style.join(";");
        styleString += ";";
        currentStyle = style;
        highlighter->setStyle(currentStyle);
        stylesColors[currentStyleName] = style;
        QFile file("styles/" + currentStyleName + ".txt");
        file.open(QIODevice::WriteOnly);
        file.write(styleString.toLocal8Bit().constData());
        file.close();
    }
    delete m;
}
/*9*/
void Textedit::loadStyle() {
```

```
    QString fn = QFileDialog::getOpenFileName(this, tr("Open File..."),
                                              QString(), tr("txt (*.txt);;All
Files (*)"));
    if (!fn.isEmpty()) {
        if (!QFile::exists(fn))
            return;
        QFile file(fn);
        if (!file.open(QFile::ReadOnly | QFile::Text)) {
            QMessageBox::warning(this, "warning", "Cannot open file" +
file.errorString());
            return;
        }

        QByteArray data = file.readAll();
        QString str = QString::fromLocal8Bit(data);
        QStringList colors = str.split(QRegExp(";"),
QString::SkipEmptyParts);
        colors = colors.filter(QRegExp("^#([A-Fa-f0-9]{6}|[A-Fa-f0-
9]{3})$"));

//        qDebug() << file.fileName();
        QFileInfo fileInfo(file.fileName());
        QAction *a;
        a = new QAction(fileInfo.baseName(), this);
        a->setCheckable(true);
        connect(a, SIGNAL(triggered()), this, SLOT(setStyle()));
        a->setObjectName(fileInfo.baseName());
//        a->isChecked(true);
        ag1->addAction(a);
        allStyles->addActions(ag1->actions());
//        qDebug() << colors;
        a->setChecked(true);
        stylesColors[fileInfo.baseName()] = colors;
        currentStyle = colors;
        currentStyleName = fileInfo.baseName();
        highlighter->setStyle(colors);
    }
}

void Textedit::setStyle() {
    currentStyle = stylesColors[QObject::sender()->objectName()];
    currentStyleName = QObject::sender()->objectName();
    highlighter->setStyle(stylesColors[QObject::sender()->objectName()]);
}

void Textedit::toggleSyntaxHighlight(bool f) {
    highlighter->setIsVisible(f);
}

void Textedit::cursorPosLabel() {
    cursorPosition->setText(textEdit->getCursorPosition());
}


void Textedit::selectWord() {
    QTextCursor cursor = textEdit->textCursor();
```

```
    cursor.select(QTextCursor::WordUnderCursor);
    textEdit->setTextCursor(cursor);
}

void Textedit::selectLine() {
    QTextCursor cursor = textEdit->textCursor();
    cursor.movePosition(QTextCursor::StartOfBlock);
    cursor.movePosition(QTextCursor::EndOfBlock, QTextCursor::KeepAnchor);
    textEdit->setTextCursor(cursor);
}

void Textedit::context(QContextMenuEvent *event) {
    Q_UNUSED(event)
    QMenu *menu = new QMenu;
    menu->addAction(undoAction);
    menu->addAction(redoAction);

    QAction* a;

    if (!textEdit->textCursor().hasSelection()) {
        a = new QAction(QString::fromUtf8("Выделить слово"), this);
        connect(a, SIGNAL(triggered()), this, SLOT(selectWord()));
        menu->addAction(a);

        a = new QAction(QString::fromUtf8("Выделить строку"), this);
        connect(a, SIGNAL(triggered()), this, SLOT(selectLine()));
        menu->addAction(a);
    }

    if (textEdit->textCursor().hasSelection()) {
        menu->addAction(copyAction);
        menu->addAction(cutAction);

        a = new QAction(tr("&Delete"), this);
        connect(a, SIGNAL(triggered()), this, SLOT(deleteText()));
        menu->addAction(a);
    }
    menu->addAction(pasteAction);
    menu->addAction(selectAllAction);
//    menu.exec(event->globalPos());
    menu->popup(QCursor::pos());
//    qDebug() << QCursor::pos();
//    menu.show();
//    qDebug() << event->globalPos();
}

void Textedit::loadStyles() {
    QDir *dir = new QDir("styles");
    dir->setFilter(QDir::Files | QDir::NoDot | QDir::NoDotDot);
    QFileInfoList list = dir->entryInfoList();
    for (int i = 0; i < list.size(); i++) {
        QFileInfo fileInfo = list.at(i);
        if(!(fileInfo.suffix() == "txt")) {
            continue;
        }
        QFile file(fileInfo.absoluteFilePath());
```

```
        if (!file.open(QFile::ReadOnly | QFile::Text)) {
            QMessageBox::warning(this, "warning", "Cannot open file" +
file.errorString());
            continue;
        }

        QByteArray data = file.readAll();
        QString str = QString::fromLocal8Bit(data);
        QStringList colors = str.split(QRegExp(";"),
QString::SkipEmptyParts);
        colors = colors.filter(QRegExp("^#([A-Fa-f0-9]{6}|[A-Fa-f0-
9]{3})$"));

        QAction *a;
        a = new QAction(fileInfo.baseName(), this);
        a->setCheckable(true);
        connect(a, SIGNAL(triggered()), this, SLOT(setStyle()));
        a->setObjectName(fileInfo.baseName());
        ag1->addAction(a);
        allStyles->addActions(ag1->actions());
//        qDebug() << colors;
        stylesColors[fileInfo.baseName()] = colors;
    }
}

void Textedit::toggleStatusBar(bool f) {
    statusBar()->setVisible(f);
}

void Textedit::updateStrings() {
    stringAmount->setText(QString::fromUtf8(" Строк: ") +
QString::number(textEdit->blockCount()) + " ");
}

void Textedit::updateChars() {
    charAmount->setText(QString::fromUtf8(" Символов: ") +
QString::number(textEdit->toPlainText().length()) + " ") ;
}

void Textedit::updateWords() {
    wordAmount->setText(QString::fromUtf8(" Слов: ") +
QString::number(textEdit->toPlainText()
                        .split(QRegExp("(\\s|\\n|\\r)+"),
                               QString::SkipEmptyParts).count()) + " ") ;
}

void Textedit::toggleToolBar(bool f) {
    tb->setVisible(f);
}

void Textedit::fileNew() {
    if (maybeSave()) {
        textEdit->clear();
        setCurrentFileName(QString());
        saveAction->setEnabled(false);
        lastSaved->setText(QString::fromUtf8(" П. сохранение: "));
```

```
        lastChanged->setText(QString::fromUtf8(" П. изменение: "));
        documentSize->setText(QString::fromUtf8(" Размер: "));
        wordAmount->setText(QString::fromUtf8(" Слов: "));
        charAmount->setText(QString::fromUtf8(" Символов: "));
        stringAmount->setText(QString::fromUtf8(" Строк: "));
    }
}

void Textedit::fileOpen() {
    if(!maybeSave())
        return;
    QString fn = QFileDialog::getOpenFileName(this, tr("Open File..."),
                                               QString(), tr("All Files
(*)"));
    if (!fn.isEmpty())
        load(fn);
}


bool Textedit::load(const QString &f)
{
    if (!QFile::exists(f))
        return false;
    QFile file(f);
    if (!file.open(QFile::ReadOnly | QFile::Text)) {
        QMessageBox::warning(this, "warning", "Cannot open file" +
file.errorString());
        return false;
    }

    QByteArray data = file.readAll();
    textEdit->setPlainText(QString::fromLocal8Bit(data));

    lastSaved->setText(QString::fromUtf8(" П. сохранение: "));
    lastChanged->setText(QString::fromUtf8(" П. изменение: "));
    setCurrentFileName(f);
    updateSize();
    updateWords();
    updateChars();
    updateStrings();
    return true;
}

void Textedit::lastSavedUpdate() {
    QTime ct = QTime::currentTime();
    lastSaved->setText(QString::fromUtf8(" П. сохранение: ") +
ct.toString(Qt::TextDate) + "   ");
//    qDebug() << ct.toString(Qt::SystemLocaleLongDate);
}

void Textedit::lastChangedUpdate() {
    QTime ct = QTime::currentTime();
    lastChanged->setText(QString::fromUtf8(" П. изменение: ") +
ct.toString(Qt::TextDate) + "   ");
}
```

```
bool Textedit::maybeSave() {
    if (!textEdit->document()->isModified())
        return true;
//    if (fileName.startsWith(QLatin1String(":/")))
//        return true;
    QMessageBox::StandardButton ret;
    ret = QMessageBox::warning(this, tr("Application"),
                               QString::fromUtf8("Документ был
модифицирован.\nСохранить изменения?"),
                               QMessageBox::Save | QMessageBox::Discard
                               | QMessageBox::Cancel);
    if (ret == QMessageBox::Save)
        return fileSave();
    else if (ret == QMessageBox::Cancel)
        return false;
    return true;
}

bool Textedit::fileSave() {
    if (fileName.isEmpty())
        return fileSaveAs();

    QTextDocumentWriter writer(fileName);
    bool success = writer.write(textEdit->document());
    if (success) {
        textEdit->document()->setModified(false);
        lastSavedUpdate();
        updateSize();
    }
    return success;
}

bool Textedit::fileSaveAs() {
    QString fn = QFileDialog::getSaveFileName(this, tr("Save as..."),
//                                              QString(), tr("ODF files
(*.odt);;HTML-Files (*.htm *.html);;All Files (*)"));
                                             QString(), tr("All Files
(*)"));
    if (fn.isEmpty())
        return false;
//    if (! (fn.endsWith(".odt", Qt::CaseInsensitive) || fn.endsWith(".htm",
Qt::CaseInsensitive) || fn.endsWith(".html", Qt::CaseInsensitive)) )
//        fn += ".odt"; // default
    setCurrentFileName(fn);
    return fileSave();
}

void Textedit::updateSize() {
    QFileInfo file(fileName);
    documentSize-> setText(QString::fromUtf8(" Размер: ") +
QString::number(file.size() / 1024.) + " kb ");
 }

void Textedit::deleteText() {
    textEdit->textCursor().removeSelectedText();
}
```

```
void Textedit::setCurrentFileName(const QString &fileName)
{
    this->fileName = fileName;
    textEdit->document()->setModified(false);

    QString shownName;
    if (fileName.isEmpty())
        shownName = "untitled.txt";
    else
        shownName = QFileInfo(fileName).fileName();

    if(shownName.size() > 32) {
        shownName.resize(32);
        shownName   += "...";
    }

    setWindowTitle(tr("%1[*]").arg(shownName));
    setWindowModified(false);
}

void Textedit::setupEditActions() {
    QMenu *menu = new QMenu(QString::fromUtf8("Правка") , this);
    menuBar()->addMenu(menu);

    undoAction = new QAction(QIcon(":images/undo.png"),
QString::fromUtf8("Отменить"), this);
    undoAction->setShortcut(QKeySequence::Undo);
    tb->addAction(undoAction);
    menu->addAction(undoAction);

    redoAction = new QAction(QIcon(":images/redo.png"),
QString::fromUtf8("Повторить"), this);
    redoAction->setPriority(QAction::LowPriority);
    redoAction->setShortcut(QKeySequence::Redo);
    tb->addAction(redoAction);
    menu->addAction(redoAction);

    cutAction = new QAction(QIcon(":images/cut.png"),
QString::fromUtf8("Вырезать"), this);
    cutAction->setPriority(QAction::LowPriority);
    cutAction->setShortcut(QKeySequence::Cut);
    tb->addAction(cutAction);
    menu->addAction(cutAction);

    copyAction = new QAction(QIcon(":images/copy.png"),
QString::fromUtf8("Копировать"), this);
    copyAction->setPriority(QAction::LowPriority);
    copyAction->setShortcut(QKeySequence::Copy);
    tb->addAction(copyAction);
    menu->addAction(copyAction);

    pasteAction = new QAction(QIcon(":images/paste.png"),
QString::fromUtf8("Вставить"), this);

    pasteAction->setPriority(QAction::LowPriority);
```

```
    pasteAction->setShortcut(QKeySequence::Paste);
    tb->addAction(pasteAction);
    menu->addAction(pasteAction);

    selectAllAction = new QAction(QString::fromUtf8("Выделить все"), this);
    selectAllAction->setPriority(QAction::LowPriority);
    selectAllAction->setShortcut(QKeySequence::SelectAll);
    menu->addAction(selectAllAction);

    findAction = new QAction(QIcon(":images/find.png"),
QString::fromUtf8("Найти"), this);
    menu->addAction(findAction);

    findAndReplaceAction = new QAction(QIcon(":images/replace.png"),
QString::fromUtf8("Найти и заменить"), this);
    menu->addAction(findAndReplaceAction);
//new-
    cutAction->setEnabled(false);
    copyAction->setEnabled(false);

    connect(cutAction, SIGNAL(triggered()), textEdit, SLOT(cut()));
    connect(copyAction, SIGNAL(triggered()), textEdit, SLOT(copy()));
    connect(pasteAction, SIGNAL(triggered()), textEdit, SLOT(paste()));
    connect(selectAllAction, SIGNAL(triggered()), textEdit,
SLOT(selectAll()));
    connect(findAction, SIGNAL(triggered()), this, SLOT(findText()));
    connect(findAndReplaceAction, SIGNAL(triggered()), this,
SLOT(findAndReplaceText()));

    connect(textEdit, SIGNAL(copyAvailable(bool)), cutAction,
SLOT(setEnabled(bool)));
    connect(textEdit, SIGNAL(copyAvailable(bool)), copyAction,
SLOT(setEnabled(bool)));
}

void Textedit::setupHelpAction() {
    QMenu *menu = new QMenu(QString::fromUtf8("Справка"), this);
    menuBar()->addMenu(menu);

    QAction* a;
    a = new QAction(QString::fromUtf8("О программе"), this);
    connect(a, SIGNAL(triggered()), this, SLOT(about()));
    menu->addAction(a);
}

void Textedit::about() {
    AboutModal *d = new AboutModal(this);
    d->exec();
    delete d;
}

void Textedit::setupTextActions() {
    QMenu *menu = new QMenu(QString::fromUtf8("Формат"), this);
    menuBar()->addMenu(menu);

    QAction* a;
```

```
    wordWrapAction = new QAction(QString::fromUtf8("Перенос по словам"),
this);
    wordWrapAction->setCheckable(true);
    wordWrapAction->setChecked(true);
    connect(wordWrapAction, SIGNAL(toggled(bool)), this,
SLOT(toggleLineWrapMode(bool)));
    menu->addAction(wordWrapAction);

    a = new QAction(QString::fromUtf8("Выбор шрифта"), this);
    connect(a, SIGNAL(triggered()), this, SLOT(ChangeFonT()));
    menu->addAction(a);

}

void Textedit::ChangeFonT/*t*/() {
    textEdit->setFont(QFontDialog::getFont(0, textEdit->font()));
}

void Textedit::findAndReplaceAccept() {
    if(!textEdit->find(m2->getFind())) {
        QMessageBox::information(this, "", QString::fromUtf8("Ничего не
найдено или найлено последнее"));
        return;
    }
    QTextCursor cursor = textEdit->textCursor();
    textEdit->setTextCursor(cursor);
//    cursor.clearSelection();
//    cursor.movePosition(QTextCursor::NextWord, QTextCursor::KeepAnchor);
    cursor.insertText(m2->getReplace());
    cursor.movePosition(QTextCursor::PreviousCharacter,
QTextCursor::KeepAnchor, m2->getReplace().size());
    textEdit->setTextCursor(cursor);
}

void Textedit::findAndReplaceText() {

    if (!m2) {
        m2 = new FindAndReplaceModal(this);
        connect(m2, SIGNAL(ok()), SLOT(findAndReplaceAccept()));
    }
    m2->show();
}

void Textedit::findText() {
    if (!m1) {
        m1 = new FindModal(this);
        connect(m1, SIGNAL(ok()), this, SLOT(findAccept()));
    }
    m1->show();
}

void Textedit::findAccept() {
    if(!textEdit->find(m1->getData())) {
        QMessageBox::information(this, "404", QString::fromUtf8("Ничего не
найдено или найлено последнее"));
```

```
    }
}


void Textedit::toggleLineWrapMode(bool f) {
    textEdit->setLineWrapMode(f ? QPlainTextEdit::WidgetWidth :
QPlainTextEdit::NoWrap);
}

void Textedit::changeBackground() {
    QColor color = QColorDialog::getColor();
    if (color.isValid()) {
        int r, g, b;
        color.getRgb(&r, &g, &b);
        backgroundColor = QString("background-color: rgb(%1, %2,
%3);").arg(r).arg(g).arg(b);
        textEdit->setStyleSheet(
                    tr("background-color: rgb(%1, %2,
%3);").arg(r).arg(g).arg(b));

    }
}

void Textedit::changeLineBackground() {
    QColor color = QColorDialog::getColor();
    if (color.isValid()) {
        textEdit->setLineBackground(color);
    }
}

void Textedit::closeEvent(QCloseEvent *event) {
//    Q_UNUSED(event);

    if(!maybeSave()) {
          event->ignore();
    }
}

void Textedit::tryClose() {
    close();
}

void Textedit::ini() {
    QSettings settings("lab12_lezhninq", QSettings::IniFormat);
    QString fileName = settings.value("fileName", QString()).toString();
    if (!QFile::exists(fileName))
        setCurrentFileName(QString());
    else
        load(fileName);

    bool wordWrap = settings.value("wordWrap", true).toBool(),
            stringNumber = settings.value("stringNumber", true).toBool(),
            highlightString = settings.value("highlightString",
true).toBool(),
            statusBar = settings.value("statusBar", true).toBool(),
            toolBar = settings.value("toolBar", true).toBool(),
```

```
            syntaxHighlight = settings.value("syntaxHighlight",
true).toBool();


    wordWrapAction->setChecked(wordWrap);
    stringNumberAction->setChecked(stringNumber);
    highlightStringAction->setChecked(highlightString);
    statusBarAction->setChecked(statusBar);
    toolBarAction->setChecked(toolBar);
    syntaxHighlightAction->setChecked(syntaxHighlight);

    QString font = settings.value("font", "").toString();
    if (font != "") {
        QFont fontt;
        fontt.fromString(font);
        textEdit->setFont(fontt);
    }

    QString lineColor = settings.value("currentLineColor",
"#e3e3e3").toString();
    QColor color;
    color.setNamedColor(lineColor);
    textEdit->setLineBackground(color);

    QString bgColor = settings.value("bgcolor", "background-color: rgb(255,
255, 255);").toString();
    textEdit->setStyleSheet(bgColor);

    int cursorPosit = settings.value("cursorPosition", 0).toInt();
    QTextCursor cursor = textEdit->textCursor();
    if (cursorPosit > textEdit->toPlainText().length()) {
        cursorPosit = textEdit->toPlainText().length();
    }
    cursor.setPosition(cursorPosit);
    textEdit->setTextCursor(cursor);
}

Textedit::~Textedit() {
    QSettings settings("lab12_lezhninq", QSettings::IniFormat);
    settings.setValue("fileName", fileName);
    settings.setValue("wordWrap", wordWrapAction->isChecked());
    settings.setValue("stringNumber", stringNumberAction->isChecked());
    settings.setValue("highlightString", highlightStringAction->isChecked());
    settings.setValue("statusBar", statusBarAction->isChecked());
    settings.setValue("toolBar", toolBarAction->isChecked());
    settings.setValue("syntaxHighlight", syntaxHighlightAction->isChecked());
    settings.setValue ("font", textEdit->font().toString());
    settings.setValue("bgcolor", backgroundColor);
    settings.setValue("currentLineColor", textEdit->getLineColor().name());
    settings.setValue("cursorPosition", textEdit->textCursor().position());
}
```

А4. Исходный код файла **editarea.h**

```
#ifndef EDITAREA_H
#define EDITAREA_H

#include <QPlainTextEdit>
#include <QPainter>
#include <QTextBlock>
#include <QDebug>
#include <QLabel>

class EditArea : public QPlainTextEdit {
    Q_OBJECT
public:
    EditArea(QWidget *parent = 0);
    void lineNumberAreaPaintEvent(QPaintEvent *);
    int lineNumberAreaWidth();
    void setLineBackground(QColor);
    QColor getLineColor();
    QString getCursorPosition();

protected:
    void resizeEvent(QResizeEvent *);
    void contextMenuEvent(QContextMenuEvent *event);

public slots:
    void toggleLineVisible(bool);

private slots:
    void updateLineNumberAreaWidth(int);
    void highlightCurrentLine();
    void updateLineNumberArea(const QRect &, int);
    void toggleNumbers(bool);
    void test();

private:
    QWidget* lineNumberArea;
    QColor lineColor;
    bool isLineAreaVisible;
    bool isLineVisible;

signals:
    void context(QContextMenuEvent*);
};

class LineNumberArea : public QWidget {
public:
    LineNumberArea(EditArea *_editArea = 0);
    QSize sizeHint() const;

protected:
    void paintEvent(QPaintEvent *);

private:
    EditArea *editArea;
};
```

```
#endif // EDITAREA_H
```

A5. Исходный код файла **editarea.cpp**

```
#include "editarea.h"

EditArea::EditArea(QWidget *parent) : QPlainTextEdit(parent) {
    lineNumberArea = new LineNumberArea(this);
    lineColor = QColor(227, 227, 227);

    isLineAreaVisible = true;
    isLineVisible = true;

    connect(this, SIGNAL(blockCountChanged(int)), this,
SLOT(updateLineNumberAreaWidth(int)));
    connect(this, SIGNAL(updateRequest(QRect,int)), this,
SLOT(updateLineNumberArea(QRect,int)));
    connect(this, SIGNAL(cursorPositionChanged()), this,
SLOT(highlightCurrentLine()));
    connect(this, SIGNAL(cursorPositionChanged()), this, SLOT(test()));

    updateLineNumberAreaWidth(0);
    highlightCurrentLine();
}

QColor EditArea::getLineColor() {
    return lineColor;
}

void EditArea::contextMenuEvent(QContextMenuEvent *event) {
    emit context(event);
}

void EditArea::test() {
}

QString EditArea::getCursorPosition() {
    auto cursor = textCursor();
    const QTextBlock block = cursor.block();
    return QString::number(block.blockNumber() + 1) + ":" +
            QString::number(cursor.positionInBlock() + 1);
}

void EditArea::lineNumberAreaPaintEvent(QPaintEvent *event) {
    QPainter painter(lineNumberArea);
    painter.fillRect(event->rect(), Qt::lightGray);


    QTextBlock block = firstVisibleBlock();
    int blockNumber = block.blockNumber();
    int top = (int)
blockBoundingGeometry(block).translated(contentOffset()).top();
    int bottom = top + (int) blockBoundingRect(block).height();

    while (block.isValid() && top <= event->rect().bottom()) {
```

```
        if (block.isVisible() && bottom >= event->rect().top()) {
            QString number = QString::number(blockNumber + 1);
            painter.setPen(Qt::black);
            painter.drawText(0, top, lineNumberArea->width(),
fontMetrics().height(),
                             Qt::AlignRight, number);
        }

        block = block.next();
        top = bottom;
        bottom = top + (int) blockBoundingRect(block).height();
        ++blockNumber;
    }
}

int EditArea::lineNumberAreaWidth() {
    int digits = 1;
    int max = qMax(1, blockCount());
    while (max >= 10) {
        max /= 10;
        ++digits;
    }

    int space = 3 + fontMetrics().width(QLatin1Char('9')) * digits;

    return space;
}

void EditArea::resizeEvent(QResizeEvent *e)
{
    QPlainTextEdit::resizeEvent(e);

    QRect cr = contentsRect();
    lineNumberArea->setGeometry(QRect(cr.left(), cr.top(),
lineNumberAreaWidth(), cr.height()));
}

void EditArea::updateLineNumberAreaWidth(int /* newBlockCount */) {
    if (!isLineAreaVisible) {
        setViewportMargins(0, 0, 0, 0);
        return;
    }
    setViewportMargins(lineNumberAreaWidth(), 0, 0, 0);
}

void EditArea::setLineBackground(QColor color) {
    lineColor = color;
    highlightCurrentLine();
}

void EditArea::toggleNumbers(bool f) {
    isLineAreaVisible = f;
    lineNumberArea->setHidden(!isLineAreaVisible);
    updateLineNumberAreaWidth(0);
}
```

```
void EditArea::toggleLineVisible(bool f) {
    isLineVisible = f;
    highlightCurrentLine();
}

void EditArea::updateLineNumberArea(const QRect &rect, int dy) {
    if (dy)
        lineNumberArea->scroll(0, dy);
    else
        lineNumberArea->update(0, rect.y(), lineNumberArea->width(),
rect.height());

    if (rect.contains(viewport()->rect()))
        updateLineNumberAreaWidth(0);
}

void EditArea::highlightCurrentLine() {
    QList<QTextEdit::ExtraSelection> extraSelections;

    if (!isReadOnly()) {
        QTextEdit::ExtraSelection selection;
        if (isLineVisible) {
            selection.format.setBackground(lineColor);
            selection.format.setProperty(QTextFormat::FullWidthSelection,
true);
            selection.cursor = textCursor();
            selection.cursor.clearSelection();
        }
        extraSelections.append(selection);
    }

    setExtraSelections(extraSelections);
}




LineNumberArea::LineNumberArea(EditArea *_editArea) : QWidget(_editArea) {
    editArea = _editArea;
}

QSize LineNumberArea::sizeHint() const {
    return QSize(editArea->lineNumberAreaWidth(), 0);
}

void LineNumberArea::paintEvent(QPaintEvent *event) {
    editArea->lineNumberAreaPaintEvent(event);
}
```

А6. Исходный код файла **syntaxhighlighter.h**

```
#ifndef HIGHLIGHTER_H
#define HIGHLIGHTER_H

#include <QSyntaxHighlighter>
#include <QHash>
#include <QTextCharFormat>
#include <QDebug>

class QTextDocument;

class Highlighter : public QSyntaxHighlighter {
    Q_OBJECT
public:
    Highlighter(QTextDocument* parent = 0);
    void setIsVisible(bool f);
    void setStyle(QStringList &style);
protected:
    void highlightBlock(const QString &text);
private:
    bool isVisible;
    struct HighlightingRule {
        QRegExp pattern;
        QTextCharFormat *format;
    };
    QVector<HighlightingRule> highlightingRules;
    QRegExp commentStartExpression;
    QRegExp commentEndExpression;
    QTextCharFormat keywordFormat;
    QTextCharFormat PreprocDirectiveFormat;
    QTextCharFormat singleLineCommentFormat;
    QTextCharFormat multiLineCommentFormat;
    QTextCharFormat quotationFormat;
    QTextCharFormat functionFormat;
    void standart();
public slots:
    void c89();
    void cpp9803();
    void cpp1114();
    void cpp17();
    void cpp20();
};

#endif // HIGHLIGHTER_H
```

А7. Исходный код файла **syntaxhighlighter.cpp**

```
#include "highlighter.h"

Highlighter::Highlighter(QTextDocument *parent) : QSyntaxHighlighter(parent)
{
    QColor color;
    isVisible = true;
    /*y*/
    color.setNamedColor("#0000ff");
```

```
    singleLineCommentFormat.setForeground(color);
    multiLineCommentFormat.setForeground(color);

    color.setNamedColor("#ff00ff");
    quotationFormat.setForeground(color);

    color.setNamedColor("#ff0000");
    functionFormat.setFontItalic(true);
    functionFormat.setForeground(color);

    color.setNamedColor("#338e5B");
    keywordFormat.setForeground(color);
    keywordFormat.setFontWeight(QFont::Bold);

    color.setNamedColor("#a020f0");
    PreprocDirectiveFormat.setForeground(color);

    commentStartExpression = QRegExp("/\\*");
    commentEndExpression = QRegExp("\\*/");

    c89();
}
/*9*/
void Highlighter::setIsVisible(bool f) {
    isVisible = f;
    rehighlight();
}

void Highlighter::standart() {

    HighlightingRule rule;

    rule.pattern = QRegExp("\\b[A-Za-z0-9_]+(?=\\()");
    rule.format = &functionFormat;
    highlightingRules.append(rule);

    rule.pattern = QRegExp("//[^\n]*");
    rule.format = &singleLineCommentFormat;
    highlightingRules.append(rule);

    rule.pattern = QRegExp("\".*\"");
    rule.format = &quotationFormat;
    highlightingRules.append(rule);

    rule.pattern = QRegExp("\'.*\'");
    rule.format = &quotationFormat;
    highlightingRules.append(rule);

    rule.pattern = QRegExp("<.*>");
    rule.format = &quotationFormat;
    highlightingRules.append(rule);
}

void Highlighter::setStyle(QStringList &style) {
    QColor color;
    color.setNamedColor(style.at(0));
```

```
    singleLineCommentFormat.setForeground(color);
    multiLineCommentFormat.setForeground(color);

    color.setNamedColor(style.at(1));
    quotationFormat.setForeground(color);

    color.setNamedColor(style.at(2));
    functionFormat.setForeground(color);

    color.setNamedColor(style.at(3));
    keywordFormat.setForeground(color);

    color.setNamedColor(style.at(4));
    PreprocDirectiveFormat.setForeground(color);

    rehighlight();
}

void Highlighter::c89() {
    highlightingRules.clear();
    HighlightingRule rule;
    QStringList keywordPatterns;
    keywordPatterns << "\\bauto\\b" << "\\bbreak\\b" << "\\bcase\\b"
                        << "\\bchar\\b" << "\\bconst\\b" << "\\bcontinue\\b"
                        << "\\bdefault\\b" << "\\bdo\\b" << "\\bdouble\\b"
                        << "\\belse\\b" << "\\benum\\b" << "\\bextern\\b"
                        << "\\bfloat\\b" << "\\bfor\\b" << "\\bgoto\\b"
                        << "\\bif\\b" << "\\bint\\b" << "\\blong\\b"
                        << "\\bregister\\b" << "\\breturn\\b" <<
"\\bshort\\b"
                        << "\\bsigned\\b" << "\\bsizeof\\b" << "\\bstatic\\b"
                        << "\\bstruct\\b" << "\\bswitch\\b" <<
"\\btypedef\\b"
                        << "\\bunion\\b" << "\\bvolatile\\b" << "\\bwhile\\b"
                        << "\\bunsigned\\b" << "\\bvoid\\b" <<
"\\bvolatile\\b";
    foreach (const QString &pattern, keywordPatterns) {
        rule.pattern = QRegExp(pattern);
        rule.format = &keywordFormat;
        highlightingRules.append(rule);
    }

    QStringList directivePatterns;
    directivePatterns << "#include" << "#define" << "#elif" <<
                         "#else" << "#endif" << "#error" <<
                         "#if" << "#ifdef" << "#ifndef" << "#line" <<
                         "#pragma" << "#undef" << "#using";
    foreach (const QString &pattern, directivePatterns) {
        rule.pattern = QRegExp(pattern);
        rule.format = &PreprocDirectiveFormat;
        highlightingRules.append(rule);
    }
    standart();
    rehighlight();
}
void Highlighter::cpp9803(){
```

```
    highlightingRules.clear();
    HighlightingRule rule;
    QStringList keywordPatterns;
    keywordPatterns << "\\bauto\\b" << "\\bbreak\\b" << "\\bcase\\b"
                    << "\\bchar\\b" << "\\bconst\\b" << "\\bcontinue\\b"
                    << "\\bdefault\\b" << "\\bdo\\b" << "\\bdouble\\b"
                    << "\\belse\\b" << "\\benum\\b" << "\\bextern\\b"
                    << "\\bfloat\\b" << "\\bfor\\b" << "\\bgoto\\b"
                    << "\\bif\\b" << "\\bint\\b" << "\\blong\\b"
                    << "\\bregister\\b" << "\\breturn\\b" << "\\bshort\\b"
                    << "\\bsigned\\b" << "\\bsizeof\\b" << "\\bstatic\\b"
                    << "\\bstruct\\b" << "\\bswitch\\b" << "\\btypedef\\b"
                    << "\\bunion\\b" << "\\bvolatile\\b" << "\\bwhile\\b"
                    << "\\bunsigned\\b" << "\\bvoid\\b" << "\\bvolatile\\b"
                    << "\\band\\b" << "\\band_eq\\b" << "\\basm\\b" <<
                       "\\bbitand\\b" << "\\bbitor\\b" << "\\bbool\\b" <<
                       "\\bcatcg\\b" << "\\bclass\\b" << "\\bcompl\\b" <<
                       "\\bconst_cast\\b" << "\\bdelete\\b" <<
"\\dynamic_cast\\b" <<
                       "\\bexplicit\\b" << "\\bexport\\b" << "\\bfalse\\b" <<
                       "\\bfriend\\b" << "\\bmutable\\b" << "\\bnamespace\\b"
   << "\\bnew\\b"  << "\\bnot\\b" << "\\bnot_eq\\b" << "\\bor\\b" <<
"\\bor_rq\\b" <<
                       "\\bprivate\\b" << "\\bprotected\\b" << "\\bpublic\\b"
<<
                       "\\breinterpret_cast\\b" << "\\bstatic_cast\\b"  <<
                       "\\btemplate\\b" << "\\bthis\\b" << "\\bthrow\\b" <<
                       "\\btrue\\b" << "\\btry\\b" << "\\btypeid\\b" <<
                       "\\btypename\\b" << "\\busing\\b" << "\\bvirtual\\b"
<<
                       "\\bwchar_t\\b" << "\\bxor\\b" << "\\bxor_eq\\b";

    foreach (const QString &pattern, keywordPatterns) {
        rule.pattern = QRegExp(pattern);
        rule.format = &keywordFormat;
        highlightingRules.append(rule);
    }
    QStringList directivePatterns;
    directivePatterns << "#include" << "#define" << "#elif" <<
                         "#else" << "#endif" << "#error" <<
                         "#if" << "#ifdef" << "#ifndef" << "#line" <<
                         "#pragma" << "#undef" << "#using";
    foreach (const QString &pattern, directivePatterns) {
        rule.pattern = QRegExp(pattern);
        rule.format = &PreprocDirectiveFormat;
        highlightingRules.append(rule);
    }
    standart();
    rehighlight();
}

//void c89();
//void cpp9803();
void Highlighter::cpp1114() {
    highlightingRules.clear();
    HighlightingRule rule;
```

```
    QStringList keywordPatterns;
    keywordPatterns << "\\bauto\\b" << "\\bbreak\\b" << "\\bcase\\b"
                    << "\\bchar\\b" << "\\bconst\\b" << "\\bcontinue\\b"
                    << "\\bdefault\\b" << "\\bdo\\b" << "\\bdouble\\b"
                    << "\\belse\\b" << "\\benum\\b" << "\\bextern\\b"
                    << "\\bfloat\\b" << "\\bfor\\b" << "\\bgoto\\b"
                    << "\\bif\\b" << "\\bint\\b" << "\\blong\\b"
                    << "\\bregister\\b" << "\\breturn\\b" << "\\bshort\\b"
                    << "\\bsigned\\b" << "\\bsizeof\\b" << "\\bstatic\\b"
                    << "\\bstruct\\b" << "\\bswitch\\b" << "\\btypedef\\b"
                    << "\\bunion\\b" << "\\bvolatile\\b" << "\\bwhile\\b"
                    << "\\bunsigned\\b" << "\\bvoid\\b" << "\\bvolatile\\b"
                    << "\\band\\b" << "\\band_eq\\b" << "\\basm\\b" <<
                       "\\bbitand\\b" << "\\bbitor\\b" << "\\bbool\\b" <<
                       "\\bcatcg\\b" << "\\bclass\\b" << "\\bcompl\\b" <<
                       "\\bconst_cast\\b" << "\\bdelete\\b" <<
"\\dynamic_cast\\b" <<
                       "\\bexplicit\\b" << "\\bexport\\b" << "\\bfalse\\b" <<
                       "\\bfriend\\b" << "\\bmutable\\b" << "\\bnamespace\\b"
   << "\\bnew\\b"  << "\\bnot\\b" << "\\bnot_eq\\b" << "\\bor\\b" <<
"\\bor_rq\\b" <<
                       "\\bprivate\\b" << "\\bprotected\\b" << "\\bpublic\\b"
<<
                       "\\breinterpret_cast\\b" << "\\bstatic_cast\\b"  <<
                       "\\btemplate\\b" << "\\bthis\\b" << "\\bthrow\\b" <<
                       "\\btrue\\b" << "\\btry\\b" << "\\btypeid\\b" <<
                       "\\btypename\\b" << "\\busing\\b" << "\\bvirtual\\b"
<<
                       "\\bwchar_t\\b" << "\\bxor\\b" << "\\bxor_eq\\b"<<
                       "\\balignas\\b" << "\\balignof\\b" << "\\bchar16_t\\b"
<<
                       "\\bchar32_t\\b" << "\\balignas\\b" <<
"\\bconstexpr\\b" <<
                       "\\bdecltype\\b" << "\\bnoexcept\\b" <<
"\\bnullptr\\b" <<
                       "\\bstatic_assert\\b" << "\\bthread_local\\b" <<
"\\bfinal\\b" <<
                       "\\boverride\\b";

    foreach (const QString &pattern, keywordPatterns) {
        rule.pattern = QRegExp(pattern);
        rule.format = &keywordFormat;
        highlightingRules.append(rule);
    }
    QStringList directivePatterns;
    directivePatterns << "#include" << "#define" << "#elif" <<
                         "#else" << "#endif" << "#error" <<
                         "#if" << "#ifdef" << "#ifndef" << "#line" <<
                         "#pragma" << "#undef" << "#using";
    foreach (const QString &pattern, directivePatterns) {
        rule.pattern = QRegExp(pattern);
        rule.format = &PreprocDirectiveFormat;
        highlightingRules.append(rule);
    }
    standart();
    rehighlight();
```

```
}

void Highlighter::cpp17() {
    highlightingRules.clear();
    HighlightingRule rule;
    QStringList keywordPatterns;
    keywordPatterns << "\\bauto\\b" << "\\bbreak\\b" << "\\bcase\\b"
                    << "\\bchar\\b" << "\\bconst\\b" << "\\bcontinue\\b"
                    << "\\bdefault\\b" << "\\bdo\\b" << "\\bdouble\\b"
                    << "\\belse\\b" << "\\benum\\b" << "\\bextern\\b"
                    << "\\bfloat\\b" << "\\bfor\\b" << "\\bgoto\\b"
                    << "\\bif\\b" << "\\bint\\b" << "\\blong\\b"
                    << "\\bregister\\b" << "\\breturn\\b" << "\\bshort\\b"
                    << "\\bsigned\\b" << "\\bsizeof\\b" << "\\bstatic\\b"
                    << "\\bstruct\\b" << "\\bswitch\\b" << "\\btypedef\\b"
                    << "\\bunion\\b" << "\\bvolatile\\b" << "\\bwhile\\b"
                    << "\\bunsigned\\b" << "\\bvoid\\b" << "\\bvolatile\\b"
                    << "\\band\\b" << "\\band_eq\\b" << "\\basm\\b" <<
                       "\\bbitand\\b" << "\\bbitor\\b" << "\\bbool\\b" <<
                       "\\bcatcg\\b" << "\\bclass\\b" << "\\bcompl\\b" <<
                       "\\bconst_cast\\b" << "\\bdelete\\b" <<
"\\dynamic_cast\\b" <<
                       "\\bexplicit\\b" << "\\bexport\\b" << "\\bfalse\\b" <<
                       "\\bfriend\\b" << "\\bmutable\\b" << "\\bnamespace\\b"
   << "\\bnew\\b"  << "\\bnot\\b" << "\\bnot_eq\\b" << "\\bor\\b" <<
"\\bor_rq\\b" <<
                       "\\bprivate\\b" << "\\bprotected\\b" << "\\bpublic\\b"
<<
                       "\\breinterpret_cast\\b" << "\\bstatic_cast\\b"  <<
                       "\\btemplate\\b" << "\\bthis\\b" << "\\bthrow\\b" <<
                       "\\btrue\\b" << "\\btry\\b" << "\\btypeid\\b" <<
                       "\\btypename\\b" << "\\busing\\b" << "\\bvirtual\\b"
<<
                       "\\bwchar_t\\b" << "\\bxor\\b" << "\\bxor_eq\\b"<<
                       "\\balignas\\b" << "\\balignof\\b" << "\\bchar16_t\\b"
<<
                       "\\bchar32_t\\b" << "\\balignas\\b" <<
"\\bconstexpr\\b" <<
                       "\\bdecltype\\b" << "\\bnoexcept\\b" <<
"\\bnullptr\\b" <<
                       "\\bstatic_assert\\b" << "\\bthread_local\\b";

    foreach (const QString &pattern, keywordPatterns) {
        rule.pattern = QRegExp(pattern);
        rule.format = &keywordFormat;
        highlightingRules.append(rule);
    }
    QStringList directivePatterns;
    directivePatterns << "#include" << "#define" << "#elif" <<
                         "#else" << "#endif" << "#error" <<
                         "#if" << "#ifdef" << "#ifndef" << "#line" <<
                         "#pragma" << "#undef" << "#using";
    foreach (const QString &pattern, directivePatterns) {
        rule.pattern = QRegExp(pattern);
        rule.format = &PreprocDirectiveFormat;
        highlightingRules.append(rule);
```

```
    }
    standart();
    rehighlight();
}

void Highlighter::cpp20() {
    highlightingRules.clear();
    HighlightingRule rule;
    QStringList keywordPatterns;
    keywordPatterns << "\\bauto\\b" << "\\bbreak\\b" << "\\bcase\\b"
                    << "\\bchar\\b" << "\\bconst\\b" << "\\bcontinue\\b"
                    << "\\bdefault\\b" << "\\bdo\\b" << "\\bdouble\\b"
                    << "\\belse\\b" << "\\benum\\b" << "\\bextern\\b"
                    << "\\bfloat\\b" << "\\bfor\\b" << "\\bgoto\\b"
                    << "\\bif\\b" << "\\bint\\b" << "\\blong\\b"
                    << "\\bregister\\b" << "\\breturn\\b" << "\\bshort\\b"
                    << "\\bsigned\\b" << "\\bsizeof\\b" << "\\bstatic\\b"
                    << "\\bstruct\\b" << "\\bswitch\\b" << "\\btypedef\\b"
                    << "\\bunion\\b" << "\\bvolatile\\b" << "\\bwhile\\b"
                    << "\\bunsigned\\b" << "\\bvoid\\b" << "\\bvolatile\\b"
                    << "\\band\\b" << "\\band_eq\\b" << "\\basm\\b" <<
                       "\\bbitand\\b" << "\\bbitor\\b" << "\\bbool\\b" <<
                       "\\bcatcg\\b" << "\\bclass\\b" << "\\bcompl\\b" <<
                       "\\bconst_cast\\b" << "\\bdelete\\b" <<
"\\dynamic_cast\\b" <<
                       "\\bexplicit\\b" << "\\bexport\\b" << "\\bfalse\\b" <<
                       "\\bfriend\\b" << "\\bmutable\\b" << "\\bnamespace\\b"
   << "\\bnew\\b"  << "\\bnot\\b" << "\\bnot_eq\\b" << "\\bor\\b" <<
"\\bor_rq\\b" <<
                       "\\bprivate\\b" << "\\bprotected\\b" << "\\bpublic\\b"
<<
                       "\\breinterpret_cast\\b" << "\\bstatic_cast\\b"  <<
                       "\\btemplate\\b" << "\\bthis\\b" << "\\bthrow\\b" <<
                       "\\btrue\\b" << "\\btry\\b" << "\\btypeid\\b" <<
                       "\\btypename\\b" << "\\busing\\b" << "\\bvirtual\\b"
<<
                       "\\bwchar_t\\b" << "\\bxor\\b" << "\\bxor_eq\\b"<<
                       "\\balignas\\b" << "\\balignof\\b" << "\\bchar16_t\\b"
<<
                       "\\bchar32_t\\b" << "\\balignas\\b" <<
"\\bconstexpr\\b" <<
                       "\\bdecltype\\b" << "\\bnoexcept\\b" <<
"\\bnullptr\\b" <<
                       "\\bstatic_assert\\b" << "\\bthread_local\\b" <<
"\\bchar8_t\\b" <<
                       "\\bconcept\\b" << "\\bconst_eval\\b" <<
"\\bconstinit\\b" <<
                       "\\bco_await\\b" << "\\bco_return\\b" <<
"\\bco_yeild\\b" <<
                       "\\brequires\\b" << "\\bthread_local\\b";

    foreach (const QString &pattern, keywordPatterns) {
        rule.pattern = QRegExp(pattern);
        rule.format = &keywordFormat;
        highlightingRules.append(rule);
    }
```

```
    QStringList directivePatterns;
    directivePatterns << "#include" << "#define" << "#elif" <<
                         "#else" << "#endif" << "#error" <<
                         "#if" << "#ifdef" << "#ifndef" << "#line" <<
                         "#pragma" << "#undef" << "#using" << "#export" <<
                         "#import" << "#module";
    foreach (const QString &pattern, directivePatterns) {
        rule.pattern = QRegExp(pattern);
        rule.format = &PreprocDirectiveFormat;
        highlightingRules.append(rule);
    }
    standart();
    rehighlight();
}
/*~*//*1*//*1*/
void Highlighter::highlightBlock(const QString &text) {
    if (!isVisible) {
        return;
    }
    foreach (const HighlightingRule &rule, highlightingRules) {
        QRegExp expression(rule.pattern);
        int index = expression.indexIn(text);
        while (index >= 0) {
            int length = expression.matchedLength();
            setFormat(index, length, *rule.format);
            index = expression.indexIn(text, index + length);
        }
    }
    setCurrentBlockState(0);

    int startIndex = 0;
    if (previousBlockState() != 1)
        startIndex = commentStartExpression.indexIn(text);

    while (startIndex >= 0) {
        int endIndex = commentEndExpression.indexIn(text, startIndex);
        int commentLength;
        if (endIndex == -1) {
            setCurrentBlockState(1);
            commentLength = text.length() - startIndex;
        } else {
            commentLength = endIndex - startIndex
                            + commentEndExpression.matchedLength();
        }
        setFormat(startIndex, commentLength, multiLineCommentFormat);
        startIndex = commentStartExpression.indexIn(text, startIndex +
commentLength);
    }
}
```

A8. Исходный код файла **aboutmodal.h**

```
#ifndef ABOUTMODAL_H
#define ABOUTMODAL_H

#include <QDialog>
```

```
#include <QLabel>
#include <QGridLayout>
#include <QPushButton>

class AboutModal : public QDialog
{
public:
    AboutModal(QWidget *parent = 0);
};

#endif // ABOUTMODAL_H
```

A9. Исходный код файла **aboutmodal.cpp**

```
#include "aboutmodal.h"

AboutModal::AboutModal(QWidget *parent) : QDialog(parent) {
    QGridLayout *l = new QGridLayout();

    QLabel *name = new QLabel(QString::fromUtf8("Дмитрий"));

    QLabel *photo = new QLabel(this);
    QPixmap pixmap(":images/qqq.png");
    photo->setPixmap(pixmap);

    QLabel *buildData = new QLabel(QString::fromUtf8("Дата сборки: ") +
__DATE__ + " " __TIME__);
    QLabel *buildQtVersion = new QLabel(QString::fromUtf8("Версия qt при
сборки: 4.8.7"));
    QLabel *execQtVersion = new QLabel(QString::fromUtf8("Версия qt при
запуске: ") + qVersion());

    QPushButton *closeBtn = new QPushButton(QString::fromUtf8("Закрыть"));
    connect(closeBtn, SIGNAL(clicked()), this, SLOT(reject()));

    l->addWidget(name, 0, 0);
    l->addWidget(photo, 1, 0);
    l->addWidget(buildData, 2, 0);
    l->addWidget(buildQtVersion, 3, 0);
    l->addWidget(execQtVersion, 4, 0);
    l->addWidget(closeBtn, 5, 0);

    setLayout(l);
}
```

A10. Исходный код файла **stylemodal.h**

```
#ifndef STYLEMODAL_H
#define STYLEMODAL_H

#include <QDialog>
#include <QLineEdit>
#include <QLabel>
#include <QGridLayout>
#include <QPushButton>
#include <QMessageBox>
```

```
#include <QColorDialog>
#include <QDebug>
//
class StyleModal : public QDialog {
    Q_OBJECT
public:
    StyleModal(QWidget *parent = 0, QStringList style = {});
    QStringList getData();
private:
    QLineEdit *comment;
    QLineEdit *quotation;
    QLineEdit *function;
    QLineEdit *keyword ;
    QLineEdit *directive;
    QPushButton *b1,
        *b2,
        *b3,
        *b4,
        *b5;
  private slots:
    void onOk();
    void changeColor();
    void onChange(QString);
};

#endif // STYLEMODAL_H
```

A11. Исходный код файла **stylemodal.cpp**

```
#include "stylemodal.h"

StyleModal::StyleModal(QWidget *parent, QStringList style) : QDialog(parent)
{
    QLabel *l1 = new QLabel(QString::fromUtf8("Комментрии: "));
    QLabel *l2 = new QLabel(QString::fromUtf8("Строки: "));
    QLabel *l3 = new QLabel(QString::fromUtf8("Функции: "));
    QLabel *l4 = new QLabel(QString::fromUtf8("Ключевые слова: "));
    QLabel *l5 = new QLabel(QString::fromUtf8("Директивы: "));
    comment = new QLineEdit;
    comment->setText(style.at(0));
    comment->setObjectName("1");
    connect(comment, SIGNAL(textChanged(QString)), this,
SLOT(onChange(QString)));

    quotation = new QLineEdit;
    quotation->setText(style.at(1));
    quotation->setObjectName("2");
    connect(quotation, SIGNAL(textChanged(QString)), this,
SLOT(onChange(QString)));

    function = new QLineEdit;
    function->setText(style.at(2));
    function->setObjectName("3");
    connect(function, SIGNAL(textChanged(QString)), this,
SLOT(onChange(QString)));
```

```
    keyword = new QLineEdit;
    keyword->setText(style.at(3));
    keyword->setObjectName("4");
    connect(keyword, SIGNAL(textChanged(QString)), this,
SLOT(onChange(QString)));

    directive = new QLineEdit;
    directive->setText(style.at(4));
    directive->setObjectName("5");
    connect(directive, SIGNAL(textChanged(QString)), this,
SLOT(onChange(QString)));

    b1 = new QPushButton("");
    b1->setObjectName("1");
    b1->setStyleSheet(QString("background: %1; border:
none;").arg(style.at(0)));
    connect(b1, SIGNAL(clicked()), this, SLOT(changeColor()));

    b2 = new QPushButton("");
    b2->setObjectName("2");
    b2->setStyleSheet(QString("background: %1; border:
none;").arg(style.at(1)));
    connect(b2, SIGNAL(clicked()), this, SLOT(changeColor()));

    b3 = new QPushButton("");
    b3->setObjectName("3");
    b3->setStyleSheet(QString("background: %1; border:
none;").arg(style.at(2)));
    connect(b3, SIGNAL(clicked()), this, SLOT(changeColor()));

    b4 = new QPushButton("");
    b4->setObjectName("4");
    b4->setStyleSheet(QString("background: %1; border:
none;").arg(style.at(3)));
    connect(b4, SIGNAL(clicked()), this, SLOT(changeColor()));

    b5 = new QPushButton("");
    b5->setObjectName("5");
    b5->setStyleSheet(QString("background: %1; border:
none;").arg(style.at(4)));
    connect(b5, SIGNAL(clicked()), this, SLOT(changeColor()));


    QGridLayout *lay = new QGridLayout;

    QPushButton *ok = new QPushButton(QString::fromUtf8("Принять"));
    QPushButton *cancel = new QPushButton(QString::fromUtf8("Отмена"));
    connect(ok, SIGNAL(clicked()), this, SLOT(onOk()));
    connect(cancel, SIGNAL(clicked()), this, SLOT(reject()));

    lay->addWidget(l1, 0, 0);
    lay->addWidget(comment, 0, 1);
    lay->addWidget(b1, 0, 2);
    lay->addWidget(l2, 1, 0);
    lay->addWidget(quotation, 1, 1);
    lay->addWidget(b2, 1, 2);
```

```
    lay->addWidget(l3, 2, 0);
    lay->addWidget(function, 2, 1);
    lay->addWidget(b3, 2, 2);
    lay->addWidget(l4, 3, 0);
    lay->addWidget(keyword, 3, 1);
    lay->addWidget(b4,3, 2);
    lay->addWidget(l5, 4, 0);
    lay->addWidget(directive, 4, 1);
    lay->addWidget(b5, 4, 2);
    lay->addWidget(ok, 5, 0);
    lay->addWidget(cancel, 5, 1);

    setLayout(lay);
}

void StyleModal::changeColor() {
    QColor color = QColorDialog::getColor();
    if (color.isValid()) {
        switch(QObject::sender()->objectName().toInt()) {
            case 1:
                comment->setText(color.name());
                b1->setStyleSheet(QString("background: %1; border:
none;").arg(color.name()));
                break;
            case 2:
                quotation->setText(color.name());
                b2->setStyleSheet(QString("background: %1; border:
none;").arg(color.name()));
                break;
            case 3:
                function->setText(color.name());
                b3->setStyleSheet(QString("background: %1; border:
none;").arg(color.name()));
                break;
            case 4:
                keyword->setText(color.name());
                b4->setStyleSheet(QString("background: %1; border:
none;").arg(color.name()));
                break;
            case 5:
                directive->setText(color.name());
                b5->setStyleSheet(QString("background: %1; border:
none;").arg(color.name()));
                break;
        }
    }
}

void StyleModal::onOk() {
    QString t1 = comment->text(), t2 = quotation->text(), t3 = function-
>text(),
            t4 = keyword->text(), t5 = directive->text();
    QRegExp r("^#([A-Fa-f0-9]{6}|[A-Fa-f0-9]{3})$");
    if (!r.exactMatch(t1) || !r.exactMatch(t2) ||
            !r.exactMatch(t3) || !r.exactMatch(t4) || !r.exactMatch(t5) ) {
```

```
        QMessageBox::information(this, "Wrond data", "Must be in hex
format");
        return;
    }
    emit accept();
//    "^#([A-Fa-f0-9]{6}|[A-Fa-f0-9]{3})$"
}

void StyleModal::onChange(QString color) {
    QRegExp r("^#([A-Fa-f0-9]{6}|[A-Fa-f0-9]{3})$");
    if (r.exactMatch(color)) {
        switch(QObject::sender()->objectName().toInt()) {
            case 1:
                b1->setStyleSheet(QString("background: %1; border:
none;").arg(color));
                break;
            case 2:
                b2->setStyleSheet(QString("background: %1; border:
none;").arg(color));
                break;
            case 3:
                b3->setStyleSheet(QString("background: %1; border:
none;").arg(color));
                break;
            case 4:
                b4->setStyleSheet(QString("background: %1; border:
none;").arg(color));
                break;
            case 5:
                b5->setStyleSheet(QString("background: %1; border:
none;").arg(color));
                break;
        }
    }
}

QStringList StyleModal::getData() {
    QStringList ret;
    ret << comment->text() << quotation->text() << function->text() <<
           keyword->text() << directive -> text();
    return ret;
}
```

v A12. Исходный код файла **findmodal.h**

```
#ifndef FINDMODAL_H
#define FINDMODAL_H

#include <QDialog>
#include <QLabel>
#include <QLineEdit>
#include <QGridLayout>
#include <QPushButton>
```

```
class FindModal : public QDialog {
Q_OBJECT
public:
    FindModal(QWidget *parent = 0);
    QString getData();
private:
    QLineEdit *le1;
signals:
    void ok();
};

#endif // FINDMODAL_H
```

A13. Исходный код файла **findmodal. cpp**

```
#include "findmodal.h"

FindModal::FindModal(QWidget *parent) : QDialog(parent) {
    QLabel *l1 = new QLabel(QString::fromUtf8("Найти: "));
    le1 = new QLineEdit;

    QGridLayout *lay = new QGridLayout;

    QPushButton *ok = new QPushButton(QString::fromUtf8("Принять"));
    QPushButton *cancel = new QPushButton(QString::fromUtf8("Отмена"));
    connect(ok, SIGNAL(clicked()), this, SIGNAL(ok()));
    connect(cancel, SIGNAL(clicked()), this, SLOT(reject()));

    lay->addWidget(l1, 0, 0);
    lay->addWidget(le1, 0, 1);
    lay->addWidget(ok, 1, 0);
    lay->addWidget(cancel, 1, 1);

    setLayout(lay);
}

QString FindModal::getData() {
    return le1->text();
}
```

A14. Исходный код файла **findandreplacemodal1.h**

```
#ifndef FINDANDREPLACEMODAL_H
#define FINDANDREPLACEMODAL_H

#include <QDialog>
#include <QLabel>
#include <QLineEdit>
#include <QGridLayout>
#include <QPushButton>

class FindAndReplaceModal : public QDialog
{
    Q_OBJECT
public:
    FindAndReplaceModal(QWidget *parent = 0);
```

```
    QString getFind() const;
    QString getReplace() const;
private:
    QLineEdit *le1;
    QLineEdit *le2;
signals:
    void ok();
};

#endif // FINDANDREPLACEMODAL_H
```

A15. Исходный код файла **findandreplacemodal1.cpp**

```
#include "findandreplacemodal.h"

FindAndReplaceModal::FindAndReplaceModal(QWidget *parent) : QDialog(parent) {
    QLabel *l1 = new QLabel(QString::fromUtf8("Найти: "));
    QLabel *l2 = new QLabel(QString::fromUtf8("Заменить на: "));
    le1 = new QLineEdit;
    le2 = new QLineEdit;

    QGridLayout *lay = new QGridLayout;

    QPushButton *ok = new QPushButton(QString::fromUtf8("Принять"));
    QPushButton *cancel = new QPushButton(QString::fromUtf8("Отмена"));
    connect(ok, SIGNAL(clicked()), this, SIGNAL(ok()));
    connect(cancel, SIGNAL(clicked()), this, SLOT(reject()));

    lay->addWidget(l1, 0, 0);
    lay->addWidget(le1, 0, 1);
    lay->addWidget(l2, 1, 0);
    lay->addWidget(le2, 1, 1);
    lay->addWidget(ok, 2, 0);
    lay->addWidget(cancel, 2, 1);

    setLayout(lay);
}

QString FindAndReplaceModal::getFind() const {
    return le1->text();
}

QString FindAndReplaceModal::getReplace() const {
    return le2->text();
}
```

A16. Исходный код файла **lb12.pro**

```
CONFIG +=   qt

HEADERS = \
    textedit.h \
    editarea.h \
    findmodal.h \
    findandreplacemodal.h \
    highlighter.h \
    stylemodal.h \
```

```
    aboutmodal.h

SOURCES = main.cpp \
    textedit.cpp \
    editarea.cpp \
    findmodal.cpp \
    findandreplacemodal.cpp \
    highlighter.cpp \
    stylemodal.cpp \
    aboutmodal.cpp

RESOURCES += \
    images.qrc
```

A17. Исходный код файла **images.qrc**

```
<RCC>
    <qresource prefix="/">
        <file>images/copy.png</file>
        <file>images/cut.png</file>
        <file>images/find.png</file>
        <file>images/new-file.png</file>
        <file>images/open-file.png</file>
        <file>images/paste.png</file>
        <file>images/redo.png</file>
        <file>images/replace.png</file>
        <file>images/save-file.png</file>
        <file>images/undo.png</file>
        <file>images/qqq.png</file>
    </qresource>
</RCC>
```