**Правительство Российской Федерации**

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

Кафедра “Компьютерная безопасность”

**ОТЧЕТ**

**К ЛАБОРАТОРНОЙ РАБОТЕ №5**

**по дисциплине**

**«Языки программирования»**

|  |  |  |
| --- | --- | --- |
| Работу выполнил студент группы СКБ201 | \_\_\_\_\_\_\_\_\_\_\_\_ | Т.В. Тур |
|  | подпись, дата |  |
|  |  |  |
| Работу проверила | \_\_\_\_\_\_\_\_\_\_\_\_ | М.Ю. Монина |
|  | подпись, дата |  |

**Содержание**

[Постановка задачи 3](#_Toc90729880)

[Основная часть 6](#_Toc90729881)

[Общая идея решения 6](#_Toc90729882)

[Решение задачи 6](#_Toc90729883)

[Конструкторы 7](#_Toc90729884)

[Методы и функционал “MainWindow” 8](#_Toc90729885)

[Методы и функционал “MenuBar” 9](#_Toc90729886)

[Методы и функционал “Widget” 9](#_Toc90729887)

[Методы и функционал “CodeEditor” 10](#_Toc90729888)

[Методы и функционал “LineNumberArea” 10](#_Toc90729889)

[Методы и функционал “Highlighter” 10](#_Toc90729890)

[Сборка CMake 10](#_Toc90729891)

[Функция *main* в файле *main.cpp* 11](#_Toc90729892)

[Приложение А – *mainwindow.h* 12](#_Toc90729893)

[Приложение Б – *codeeditor.h* 16](#_Toc90729894)

[Приложение В – *highlighter.h* 18](#_Toc90729895)

[Приложение Г – main.cpp 20](#_Toc90729896)

[Приложение Д – CMakeLists.txt 21](#_Toc90729897)

[Приложение Е – resource\_file.qrc 23](#_Toc90729898)

[Приложение Ж – UML 2.0 диаграмма классов 24](#_Toc90729899)

[Приложение З – mainwindow.cpp 29](#_Toc90729900)

[Приложение И – codeeditor.cpp 48](#_Toc90729901)

[Приложение К – highlighter.cpp 51](#_Toc90729902)

# 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): Си (стандарт 2011), Си++ (стандарт 2014)

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

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

1. Файл [кнопки]
   1. Новый
   2. Открыть
   3. Сохранить
   4. Сохранить как
   5. Выход
2. Правка [кнопки]
   1. Отменить
   2. Повторить
   3. Копировать
   4. Вырезать
   5. Вставить
   6. Найти
   7. Найти и заменить
   8. Выделить все
3. Формат
   1. Перенос по словам [галочка]
   2. Выбор шрифта – открывается модальный диалог выбора шрифта
4. Вид [галочки, где не указано иное]
   1. Выбор цвета фона [кнопка] – открывает модальное окно выбора цвета
   2. Выбор цвета текущей строки [кнопка] – открывает модальное окно выбора цвета
   3. Вкл/Выкл отображения нумерации строк
   4. Вкл/Выкл отображения панели инструментов
   5. Вкл/Выкл отображения строки состояния
   6. Вкл/Выкл подсветки синтаксиса
   7. Выбор синтаксиса (для подсветки) [дочернее меню] – доступно всегда, один синтаксис в дочернем меню выбран всегда
   8. Выбор/Редактирование стиля подсветки [дочернее меню] – для текущего синтаксиса, по умолчанию выбран *Default*
      1. Изменить [кнопка] – измененный стиль сохраняется в файл, имя файла становится именем стиля, стиль становится активным
      2. Загрузка стиля из файла [кнопка] – имя файла становится именем стиля, стиль становится активным
      3. Обязательная кнопка *Default*
      4. ***доступные стили [кнопки – перечисляются все стили, которые были обнаружены]***
5. Справка
   1. 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.

Реализовать подсветку синтаксиса Си (стандарт 2018), Си++17, Си++20

# Основная часть

## Общая идея решения

Разработать 5 классов решающих те или иные поставленные задачи. Классы будут содержать открытые и закрытые поля. Классу отводится своя задача, которую решает конкретно он.

## Решение задачи

Для решения задачи было разработано 7 классов, а не 5, из-за некоторых изначально закрытых возможностей, а так же одна структура: “MainWindow”, “MenuBar” ,“Widget”, “CodeEditor”, “LineNumberArea”, “Style“ и “Highlighter”.

Класс “MainWindow” наслудется от “QMainWindow”, имеет 56 закрытых полей: 7 для элементов меню типа “QMenu\*” и 31 их элементов “QAction\*”; 1 диалог стиля “QDialog\*” и 8 его кнопок “QPushButton\*”; словарь “QMap <QString, Style>”, строящийся по соответствию «имя» - «стиль» для всех доступных стилей, текущий стиль “Style”; таймер “QTimer\*” и 2 “bool” изменения дня; 3 “ QLabel\*” строки состояния; “Widget\*”, указатель но текстовую зону.

Класс “MenuBar” наследуется от “QMenuBar” и не имеет каких-либо полей.

Класс “Widget” наследуется от “QWidget” и имеет 18 закрытых полей: “MenuBar\*” панели инструментов, его подменю “QMenu\*” и 10 “QAction\*”; редактор “CodeEditor\*”, выделитель синтаксиса “Highlighter\*”, родительское окно “MainWindow\*”; 2 “QLineEdit\*” для поиска и замены; 1 “bool” для отслеживания изменений. А также 6 открытых полей: по 2 “QTime” и “QDate” для отслеживания строки состояния, текст в виде документа “QTextDocument \*” и имя текущего файла “ QString”.

Класс “CodeEditor” наследуется от “QPlainTextEdit” и имеет 3 закрытых поля: “QWidget \*” зоны нумерации строк и 2 “QColor” цвета по умолчанию. Также содержит 1 открытое поле: “bool” флаг отображения.

Класс “LineNumberArea” наследуется от “QWidget” и имеет 1 закрытое поле: “CodeEditor \*” изначального текстового поля.

Тип данных Style содержит в себе 9 “QTextCharFormat” для различных паттернов выделения.

Класс “Highlighter” наследуется от “QSyntaxHighlighter” и имеет 7 закрытых полей: тип “HighlightingRule” для правил выделения соответствующих паттернов; “QList<HighlightingRule>” выделения всех паттернов в соответствующем стиле, кроме двух паттернов “QRegularExpression” для комментариев; 2 “QStringList” для команд препроцессора и ключевых слов; состояние видимости “bool”; текущий стиль “Style” и его название “QString”.

## Конструкторы

Конструктор “MainWindow” от одного “QWidget \*” создает главное окно, согласно конструктору “QMainWindow”, задает таймер, диалоговое окно выбора стиля, поиск стилей, задает строку состояния, все меню, а также основное поле программы – “Widget”.

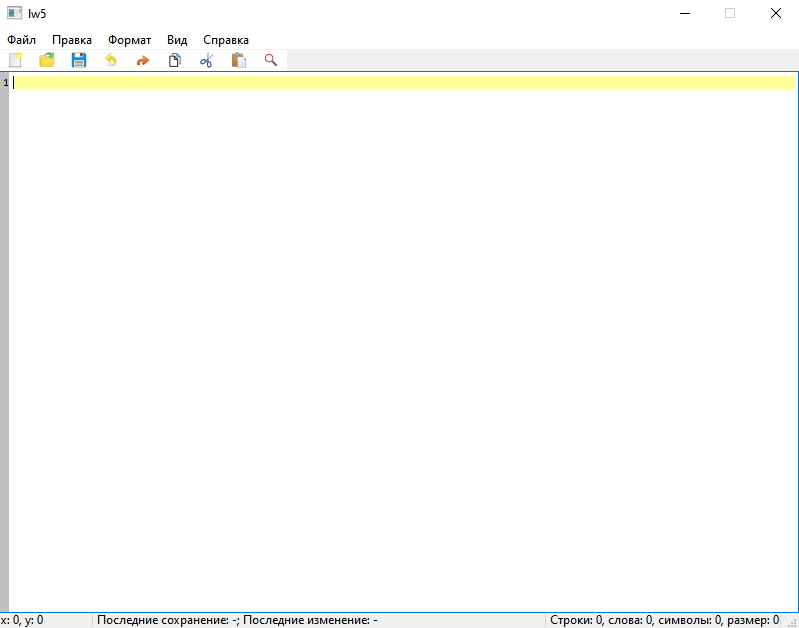


Рисунок 1 - Главное окно

Конструктор “MenuBar” от “QWidget\*” выполняет конструктор “QWidget” поумолчанию для родительского окна.

Конструктор “Widget” от “QWidget\*” и “int” задает панель инструментов и основной текстовый редактор программы.

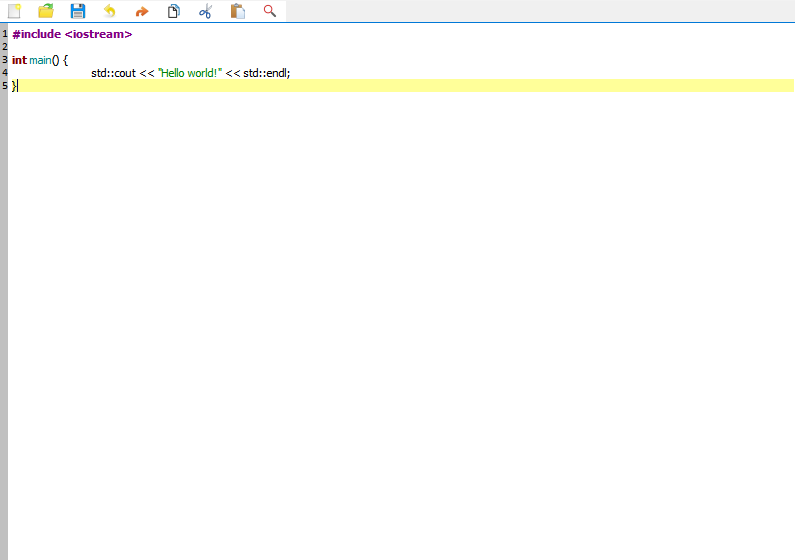


Рисунок 2. Диалоговое окно изменения поворота фигуры

Конструктор “CodeEditor” от одного “QWidget \*” создает именно текстовый элемент программы.

Конструктор “LineNumberArea” от одного “CodeEditor\*” создает столбец нумерации строк.

Конструктор “Highlighter” от одного “QTextDocument\*” задает все правила выделения, как и выделяет нужный синтаксис.

## Методы и функционал “MainWindow”

Помимо унаследованного от “QMainWindow”, “MainWidow” имеет 3 открытых слота: *updateStatus* - обновление строки состояния, *daySaved* - функция задания условия сохранения сегодня, *selectC* - выделение определенного синтаксиса в меню.

Кнопки подменю “Файл” и “Правка” выполнены в соответствии с подразумевающимся под ними функционалом. Где возможно были добавлены сочетания клавиш, выполняющие те же функции. “Формат”→”Выбор шрифта”, “Вид”→”Выбор цвета фона”, “Вид”→”Выбор цвета текущей строки”, “Вид”→”Выбор/Редактирование стиля подсветки”→”Изменить” и “Загрузка стиля из файла” открывают соответствующие модальные окна. По умолчанию все отображения, перенос строки и выделения синтаксиса включены, задается синтаксис “Си 2011” в стиле “Default”. Единовременно может быть выбран только 1 стиль и 1 синтаксис. “Справка”→”О программе” открывает модальное окно моих страданий.

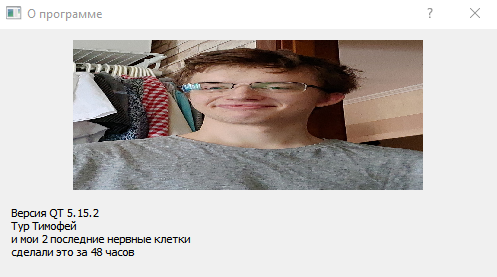


Рисунок 3. “Справка”→”О программе”

## Методы и функционал “MenuBar”

Помимо унаследованного от “QMenuBar”, “MenuBar” имеет перегрузку обработчика событий, созданную исключительно для вывода подсказок функционала панели инструментов (изначально “QMenuBar” такого не поддерживает ни в каком виде).

## Методы и функционал “Widget”

Помимо унаследованного от “QWidget”, “Widget” имеет 7 геттеров: *getX*, возвращающий координату столбца в текстовом редакторе, *getY* координаты строки, *lineCount*, *wordCount* и *symbCount* возвращают количество строк, слов, символов соответственно в редакторе в момент вызова, *kbCount* возвращает вес текущего файла в килобайтах, *getStyle* возвращает текущий стиль выделения. Имеет 1 сеттер: *setStyle*, устанавливающий стиль подсветки. Имеет 28 открытых слотов: *newFile*, *openFile*, *saveFile*, *saveFileAs*, *undo*, *redo*, *copy*, *cut*, *paste*, *find*, *findNext*, *findPrev*, *findAndReplace*, *replace*, *selectAll*, *switchWordWrap*, *showChangeFont*, *showChangeBackground*, *showChangeLineColor*, *switchNumeration*, *switchToolBar*, *switchHighlight*, *c11Syntax*, *c18Syntax*, *cpp14Syntax*, *cpp17Syntax*, *cpp20Syntax*, *showAbout*, реализующих соответствующий им функционал.

## Методы и функционал “CodeEditor”

Помимо унаследованного от “QPlainTextEdit”, “CodeEditor” имеет 4 геттера: *getBGColor* и *getLColor*, возвращающие цвета фона и линий соответственно, *lineNumberAreaWidth* возвращает ширину столбца нумерации строк, *isVisibleLineNumberArea*, возвращает видима ли строка нумерации в момент. Содержит перегрузку метода отрисовки элемента *lineNumberAreaPaintEvent* и изменения размера элемента *resizeEvent.* Имеет 5 открытых слотов: *updateLineNumberAreaWidth*, *highlightCurrentLine*, *updateLineNumberArea*, *setBackgroundColor*, *setLineColor*, реализующих соответствующий им функционал.

## Методы и функционал “LineNumberArea”

Помимо унаследованного от “QWidget”, “LineNumberArea” перегружает функцию отрисовки элемента *paintEvent*  и имеет геттер *sizeHint,* возвращающий ширину строки.

## Методы и функционал “Highlighter”

Помимо унаследованного от “QSyntaxHighlighter”, “Highlighter” перегружает функции *highlightBlock* для выделения нужных элементов. Имеет 3 геттера: *getVisible*, *getStandart* и *getStyle*, возвращающих видимость выделения, стандарт выделения и стиль выделения соответственно. Имеет 4 открытых слота: *setVisibility*, *setStandart*, *setStyle* и *setDefStyle*, являющихся сеттерами видимости, стандарта, стиля и стиля по умолчанию соответственно.

## Сборка CMake

Проект компилируется с помощью системы сборки CMake. Файл конфигурации сгенерирован IDE “Qt Creator”. В нем установлен нужный компилятор и его версия, установлены флаги для компиляции. Файл конфигурации CMake смотреть в Приложении Д.

## Функция *main* в файле *main.cpp*

В функции *main* приведен пример создания разработанного окна MainWindow. Код с функцией *main* приведен в приложении Г.

# Приложение А – *mainwindow.h*

#ifndef MAINWINDOW\_H

#define MAINWINDOW\_H

#include "codeeditor.h"

#include "highlighter.h"

#include <QMainWindow>

#include <QWidget>

#include <QMenuBar>

#include <QTextEdit>

#include <QLineEdit>

#include <QLabel>

#include <QTime>

#include <QDate>

#include <QPushButton>

#include <QSettings>

class Widget;

class MainWindow : public QMainWindow

{

Q\_OBJECT

public:

MainWindow(QWidget \*parent = nullptr);

public slots:

void updateStatus();

void daySaved();

void selectC(QString);

private slots:

void passDay();

void chooseStyle(QAction\*);

void showChangeStyle();

void showLoadStyle();

void setStyleColor();

void saveStyle();

private:

void createEditStyleDialog();

void readStyle(QSettings\*);

void writeStyle(QSettings\*);

QMenu\* \_MenuFile;

QAction\* \_ActionNew; // button ☺

QAction\* \_ActionOpen; // dialog ☺

QAction\* \_ActionSave; // dialog ☺

QAction\* \_ActionSaveAs; // dialog ☺

QAction\* \_ActionQuit; // button ☺

QMenu\* \_MenuEdit;

QAction\* \_ActionUndo; // button ☺

QAction\* \_ActionRedo; // button ☺

QAction\* \_ActionCopy; // button ☺

QAction\* \_ActionCut; // button ☺

QAction\* \_ActionPaste; // button ☺

QAction\* \_ActionFind; // nonmodal ☺

QAction\* \_ActionFindReplace; // nonmodal ☺

QAction\* \_ActionSelectAll; // button ☺

QMenu\* \_MenuFormat;

QAction\* \_ActionWordWrap; // check ☺

QAction\* \_ActionChangeFont; // modal ☺

QMenu\* \_MenuView;

QAction\* \_ActionChangeBackground; // modal ☺

QAction\* \_ActionChangeLineColor; // modal ☺

QAction\* \_ActionChangeNumeration; // check ☺

QAction\* \_ActionChangeToolBar; // check ☺

QAction\* \_ActionChangeState; // check ☺

QAction\* \_ActionChangeHighlight; // check ☺

QMenu\* \_MenuChangeSyntax; // submenu

QAction\* \_C11; // button ☺

QAction\* \_C18; // button ☺

QAction\* \_Cpp14; // button ☺

QAction\* \_Cpp17; // button ☺

QAction\* \_Cpp20; // button ☺

QMenu\* \_MenuChangeStyle; // submenu

QAction\* \_ActionChangeStyle; // modal ☺

QAction\* \_ActionLoadStyle; // modal ☺

QAction\* \_ActionDefaultStyle; // modal ☺

QMenu\* \_MenuReference;

QAction\* \_ActionAbout; // ☺

QPushButton\* \_KeywordB,

\* \_SLCommentB,

\* \_MLCommentB,

\* \_QuotationB,

\* \_SCharB,

\* \_FunctionB,

\* \_DirectiveB,

\* \_AngleB;

QDialog \*\_StyleDialog;

Widget\* \_Widget;

QLabel\* \_Cursor;

QLabel\* \_Last;

QLabel\* \_Amounts;

QTimer\* \_Timer;

bool \_DayPassedC = false,

\_DayPassedS = false;

QMap <QString, Style> \_Styles;

Style \_CurrStyle;

QAction\* \_CurrStyleAction;

};

// потому что нет правильной реализации tooltip лол

class MenuBar: public QMenuBar {

public:

explicit MenuBar(QWidget \*parent = nullptr);

protected:

virtual bool event(QEvent \*e) override;

};

class Widget : public QWidget {

Q\_OBJECT

public:

Widget(MainWindow \*parent = nullptr, int dh = 0);

long getX();

long getY();

long lineCount();

long wordCount();

long symbCount();

long kbCount();

Style getStyle();

void setStyle(Style);

QTextDocument \*\_Doc;

QTime \_LSave,

\_LChange;

QDate \_LSaveDate,

\_LChangeDate;

QString \_File;

public slots:

void newFile();

void openFile();

void saveFile();

void saveFileAs();

void undo();

void redo();

void copy();

void cut();

void paste();

void find();

void findNext();

void findPrev();

void findAndReplace();

void replace();

void selectAll();

void switchWordWrap(bool);

void showChangeFont();

void showChangeBackground();

void showChangeLineColor();

void switchNumeration(bool);

void switchToolBar(bool);

void switchHighlight(bool);

void c11Syntax();

void c18Syntax();

void cpp14Syntax();

void cpp17Syntax();

void cpp20Syntax();

void showAbout(); // ֎

private:

MenuBar\* \_ToolBar;

QAction\* \_ToolNew;

QAction\* \_ToolOpen;

QAction\* \_ToolSave;

QAction\* \_ToolUndo;

QAction\* \_ToolRedo;

QAction\* \_ToolCopy;

QAction\* \_ToolCut;

QAction\* \_ToolPaste;

QMenu\* \_ToolMenuFind; // submenu

QAction\* \_ToolFind; // nonmodal

QAction\* \_ToolReplace; // nonmodal

CodeEditor\* \_TextField;

Highlighter\* \_Highlighter;

QLineEdit\* \_FindW;

QLineEdit\* \_ReplaceW;

bool \_Changed;

MainWindow\* \_MainWindow;

};

#endif // MAINWINDOW\_H

# Приложение Б – *codeeditor.h*

#ifndef CODEEDITOR\_H

#define CODEEDITOR\_H

#include <QPlainTextEdit>

class CodeEditor : public QPlainTextEdit

{

Q\_OBJECT

public:

CodeEditor(QWidget \*parent = 0);

void lineNumberAreaPaintEvent(QPaintEvent\*);

int lineNumberAreaWidth();

QColor getBGColor() const;

QColor getLColor() const;

bool isVisibleLineNumberArea() const;

bool flag = 1;

protected:

void resizeEvent(QResizeEvent\*);

public slots:

void updateLineNumberAreaWidth(int);

void highlightCurrentLine();

void updateLineNumberArea(const QRect &, int);

void setBackgroundColor(QColor newColor);

void setLineColor(QColor newColor);

private:

QWidget \*lineNumberArea;

QColor BGColor = Qt::white,

LColor = Qt::yellow;

};

class LineNumberArea : public QWidget

{

public:

LineNumberArea(CodeEditor \*editor = 0);

QSize sizeHint() const;

protected:

void paintEvent(QPaintEvent \*event);

private:

CodeEditor \*codeEditor;

};

#endif // CODEEDITOR\_H

# Приложение В – *highlighter.h*

#ifndef HIGHLIGHTER\_H

#define HIGHLIGHTER\_H

#include <QSyntaxHighlighter>

#include <QRegularExpression>

#include <QTextCharFormat>

#include <QStringList>

struct Style {

QTextCharFormat singleLineCommentFormat,

directiveFormat,

keywordFormat,

angleFormat,

classFormat,

singleCharFormat,

quotationFormat,

functionFormat,

multiLineCommentFormat;

};

class Highlighter : public QSyntaxHighlighter

{

Q\_OBJECT

public:

Highlighter(QTextDocument \*parent = nullptr);

bool getVisible() const;

QString getStandart() const;

Style getStyle() const;

public slots:

void setVisibility(bool);

void setStandart(QString);

void setDefStyle(bool);

void setStyle(Style);

protected:

void highlightBlock(const QString &text) override;

private:

void setRules();

struct HighlightingRule

{

QRegularExpression pattern;

QTextCharFormat format;

};

QList<HighlightingRule> \_Rules;

QRegularExpression comesspres, comespres;

QStringList \_DiPatt, \_KeyPatt;

bool \_Visib = true;

Style \_Style;

QString \_S;

};

#endif // HIGHLIGHTER\_H

# Приложение Г – main.cpp

#include "mainwindow.h"

#include <QApplication>

int main(int argc, char \*argv[])

{

QApplication a(argc, argv);

MainWindow w;

w.show();

return a.exec();

}

# Приложение Д – CMakeLists.txt

cmake\_minimum\_required(VERSION 3.5)

project(lw5 VERSION 0.1 LANGUAGES CXX)

set(CMAKE\_INCLUDE\_CURRENT\_DIR ON)

set(CMAKE\_AUTOUIC ON)

set(CMAKE\_AUTOMOC ON)

set(CMAKE\_AUTORCC ON)

set(CMAKE\_CXX\_STANDARD 11)

set(CMAKE\_CXX\_STANDARD\_REQUIRED ON)

find\_package(QT NAMES Qt6 Qt5 COMPONENTS Widgets REQUIRED)

find\_package(Qt${QT\_VERSION\_MAJOR} COMPONENTS Widgets REQUIRED)

set(PROJECT\_SOURCES

main.cpp

mainwindow.cpp

mainwindow.h

codeeditor.h

codeeditor.cpp

highlighter.h

highlighter.cpp

resource\_file.qrc

)

if(${QT\_VERSION\_MAJOR} GREATER\_EQUAL 6)

qt\_add\_executable(lw5

MANUAL\_FINALIZATION

${PROJECT\_SOURCES}

)

# Define target properties for Android with Qt 6 as:

# set\_property(TARGET lw5 APPEND PROPERTY QT\_ANDROID\_PACKAGE\_SOURCE\_DIR

# ${CMAKE\_CURRENT\_SOURCE\_DIR}/android)

# For more information, see https://doc.qt.io/qt-6/qt-add-executable.html#target-creation

else()

if(ANDROID)

add\_library(lw5 SHARED

${PROJECT\_SOURCES}

)

# Define properties for Android with Qt 5 after find\_package() calls as:

# set(ANDROID\_PACKAGE\_SOURCE\_DIR "${CMAKE\_CURRENT\_SOURCE\_DIR}/android")

else()

add\_executable(lw5

${PROJECT\_SOURCES}

)

endif()

endif()

target\_link\_libraries(lw5 PRIVATE Qt${QT\_VERSION\_MAJOR}::Widgets)

set\_target\_properties(lw5 PROPERTIES

MACOSX\_BUNDLE\_GUI\_IDENTIFIER my.example.com

MACOSX\_BUNDLE\_BUNDLE\_VERSION ${PROJECT\_VERSION}

MACOSX\_BUNDLE\_SHORT\_VERSION\_STRING ${PROJECT\_VERSION\_MAJOR}.${PROJECT\_VERSION\_MINOR}

)

if(QT\_VERSION\_MAJOR EQUAL 6)

qt\_finalize\_executable(lw5)

endif()

# Приложение Е – resource\_file.qrc

<!DOCTYPE RCC><RCC version="1.0">

<qresource>

<file>images/new.png</file>

<file>images/open.png</file>

<file>images/save.png</file>

<file>images/copy.png</file>

<file>images/cut.png</file>

<file>images/paste.png</file>

<file>images/undo.png</file>

<file>images/redo.png</file>

<file>images/find.jpg</file>

<file>images/mememememe.jpg</file>

</qresource>

</RCC>

# Приложение Ж – UML 2.0 диаграмма классов

|  |  |  |
| --- | --- | --- |
| MainWindow |  | QMainWindow |
| - \_MenuFile : QMenu\*  - \_MenuEdit : QMenu\*  - \_MenuFormat : QMenu\*  - \_MenuView : QMenu\*  - \_MenuChangeSyntax : QMenu\*  - \_MenuChangeStyle : QMenu\*  - \_MenuReference : QMenu\*  - \_ActionNew : QAction\*  - \_ActionOpen : QAction\*  - \_ActionSave : QAction\*  - \_ActionSaveAs : QAction\*  - \_ActionUndo : QAction\*  - \_ActionRedo : QAction\*  - \_ActionCopy : QAction\*  - \_ActionCut : QAction\*  - \_ActionPaste : QAction\*  - \_ActionFind : QAction\*  - \_ActionFindReplace : QAction\*  - \_ActionSelectAll : QAction\*  - \_ActionWordWrap : QAction\*  - \_ActionChangeFont : QAction\*  - \_ActionChangeBackground : QAction\*  - \_ActionChangeLineColor : QAction\*  - \_ActionChangeNumeration : QAction\*  - \_ActionChangeToolBar : QAction\*  - \_ActionChangeState : QAction\*  - \_ActionChangeHighlight : QAction\*  - \_C11 : QAction\*  - \_C18 : QAction\*  - \_C++14 : QAction\*  - \_C++17 : QAction\*  - \_C++20 : QAction\*  - \_ActionChangeStyle : QAction\*  - \_ActionLoadStyle : QAction\*  - \_ActionDefaultStyle : QAction\*  - \_ActionAbout : QAction\*  - \_StyleDialog : QDialog\*  - \_KeywordB : QPushButton\*  - \_SLCommentB : QPushButton\*  - \_MLCommentB : QPushButton\*  - \_SCharB : QPushButton\*  - \_FunctionB : QPushButton\*  - \_DirectiveB : QPushButton\*  - \_AngleB : QPushButton\*  - \_Styles : QMap <QString, Style>  - \_CurrStyle : Style  - \_CurrStyleAction : QAction\*  - \_Timer : QTimer\*  - \_DayPassedC : bool  - \_DayPassedS : bool  - \_Cursor : QLabel\*  - \_Last : QLabel\*  - \_Amounts : QLabel\*  - \_Widget : Widget\* | public |  |
| Style |
|  |
|  |
|  |
|  |
|  |
|  |
| Widget |
| + MainWindow(QWidget \*parent)  + updateStatus() : void  + daySaved() : void  + selectC(QString) : void |  |  |

|  |  |  |
| --- | --- | --- |
| MenuBar |  | QMenuBar |
|  | public |  |
| + MenuBar (QWidget \*parent)  # event(QEvent \*e) : bool |  |  |

|  |  |  |
| --- | --- | --- |
| Widget |  | QWidget |
| + \_Doc : QTextDocument\*  + \_LSave : QTime  + \_LChange : QTime  + \_LSaveDate : QDate  + \_LChangeDate : QDate  + \_File : QString  - \_ToolBar : MenuBar\*  - \_ToolNew : QAction\*  - \_ToolOpen : QAction\*  - \_ToolSave : QAction\*  - \_ToolUndo : QAction\*  - \_ToolRedo : QAction\*  - \_ToolCopy : QAction\*  - \_ToolCut : QAction\*  - \_ToolPaste : QAction\*  - \_ToolMenuFind : QMenu\*  - \_ToolFind : QAction\*  - \_ToolReplace : QAction\*  - \_TextField : CodeEditor\*  - \_Highlighter : Highlighter\*  - \_FindW : QLineEdit\*  - \_ReplaceW : QLineEdit\*  - \_Changed : bool  - \_MainWindow : MainWindow\* | public |  |
| MenuBar |
|  |
|  |
|  |
|  |
| CodeEditor |
| Highlighter |
|  |
|  |
|  |
| MainWindow |
| + Widget(MainWindow \*parent, int)  + getX() : long  + getY() : long  + lineCount() : long  + wordCount() : long  + symbCount() : long  + kbCount() : long  + getStyle() : Style  + setStyle(Style) : void  + newFile() : void  + openFile() : void  + saveFile() : void  + saveFileAs() : void  + undo() : void  + redo() : void  + copy() : void  + cut() : void  + paste() : void  + find() : void  + findNext() : void  + findPrev() : void  + findAndReplace() : void  + replace() : void  + selectAll() : void  + switchWordWrap(bool) : void  + showChangeFont() : void  + showChangeBackground() : void  + showChangeLineColor() : void  + switchNumeration(bool) : void  + switchToolBar(bool) : void  + switchHighlight(bool) : void  + c11Syntax() : void  + c18Syntax() : void  + cpp14Syntax() : void  + cpp17Syntax() : void  + cpp20Syntax() : void  + showAbout() : void |  |  |

|  |  |  |
| --- | --- | --- |
| CodeEditor |  | QPlainTextEdit |
| + flag : bool  - lineNumberArea : QWidget\*  - BGColor : QColor  - LColor : QColor | public |  |
|  |
| + CodeEditor(QWidget \*parent)  + lineNumberAreaPaintEvent  (QPaintEvent\*) : void  + lineNumberAreaWidth() : int  + getBGColor() : QColor  + getLColor() : QColor  + isVisibleLineNumberArea() : bool  + updateLineNumberAreaWidth(int) : void  + highlightCurrentLine() : void  + updateLineNumberArea  (const QRect &, int) : void  + setBackgroundColor  (QColor newColor) : void  + setLineColor(QColor newColor) : void  # resizeEvent(QResizeEvent\*) : void |  |  |

|  |  |  |
| --- | --- | --- |
| LineNumberArea |  | QWidget |
| - codeEditor : CodeEditor\* | public |  |
|  |
| CodeEditor |
| + LineNumberArea(CodeEditor \*editor)  + sizeHint() : QSize  # paintEvent(QPaintEvent\*): void |  |  |

|  |  |  |
| --- | --- | --- |
| Style |  |  |
| + singleLineCommentFormat :  QTextCharFormat  + directiveFormat : QTextCharFormat  + keywordFormat : QTextCharFormat  + angleFormat : QTextCharFormat  + classFormat : QTextCharFormat  + singleCharFormat : QTextCharFormat  + quotationFormat : QTextCharFormat  + functionFormat : QTextCharFormat  + multiLineCommentFormat :  QTextCharFormat |  |  |
|  |  |  |

|  |  |  |
| --- | --- | --- |
| Highlighter |  | QSyntaxHighlighter |
| - HighlightingRule :  QRegularExpression,  QTextCharFormat  - \_Rules : QList<HighlightingRule>  - comesspres : QRegularExpression  - comespres : QRegularExpression  - \_DiPatt : QStringList  - \_KeyPatt : QStringList  - \_Visib : bool  - \_Style : Style  - \_S : QString | public |  |
|  |
|  |
|  |
| Style |
|  |
| + Highlighter(QTextDocument \*parent)  + getVisible() : bool  + getStandart() : QString  + getStyle() : Style  + setVisibility(bool) : void  + setStandart(QString) : void  + setDefStyle(bool) : void  + setStyle(Style) : void  # highlightBlock(const QString &) : void  - setRules() : void |  |  |

Приложение З – mainwindow.cpp

#include "mainwindow.h"

#include <QMenuBar>

#include <QStatusBar>

#include <QToolBar>

#include <QEvent>

#include <QHelpEvent>

#include <QToolTip>

#include <QCoreApplication>

#include <QFileDialog>

#include <QMessageBox>

#include <QTextStream>

#include <QLineEdit>

#include <QLabel>

#include <QPushButton>

#include <QGridLayout>

#include <QPlainTextEdit>

#include <QFontDialog>

#include <QColorDialog>

#include <QMimeDatabase>

#include <QTextDocument>

#include <QTimer>

#include <QFormLayout>

#include <QTextCodec>

MainWindow::MainWindow(QWidget \*parent)

: QMainWindow(parent)

{

\_Timer = new QTimer();

\_Timer->setSingleShot(true);

\_Timer->start((((24-QTime::currentTime().hour())\*60

-QTime::currentTime().minute())\*60

-QTime::currentTime().second())\*1000);

connect(\_Timer, SIGNAL(timeout()), this, SLOT(passDay()));

setFixedSize(800,600);

statusBar();

\_Widget = new Widget(this,menuBar()->height());

setCentralWidget(\_Widget);

createEditStyleDialog();

\_MenuFile = menuBar()->addMenu(tr("&Файл"));

\_ActionNew = \_MenuFile->addAction(tr("Новый"),

\_Widget, SLOT(newFile()),

QKeySequence::New);

\_ActionOpen = \_MenuFile->addAction(tr("Открыть"),

\_Widget, SLOT(openFile()),

QKeySequence::Open);

\_ActionSave = \_MenuFile->addAction(tr("Сохранить"),

\_Widget, SLOT(saveFile()),

QKeySequence::Save);

\_ActionSaveAs = \_MenuFile->addAction(tr("Сохранить как"),

\_Widget, SLOT(saveFileAs()),

QKeySequence::SaveAs);

\_ActionQuit = \_MenuFile->addAction(tr("&Выход"),

QCoreApplication::instance(),

SLOT(quit()),

QKeySequence::Quit);

\_MenuEdit = menuBar()->addMenu(tr("&Правка"));

\_ActionUndo = \_MenuEdit->addAction(tr("Отменить"),

\_Widget, SLOT(undo()),

QKeySequence::Undo);

\_ActionRedo = \_MenuEdit->addAction(tr("Повторить"),

\_Widget, SLOT(redo()),

QKeySequence::Redo);

\_ActionCopy = \_MenuEdit->addAction(tr("Копировать"),

\_Widget, SLOT(copy()),

QKeySequence::Copy);

\_ActionCut = \_MenuEdit->addAction(tr("Вырезать"),

\_Widget, SLOT(cut()),

QKeySequence::Cut);

\_ActionPaste = \_MenuEdit->addAction(tr("Вставить"),

\_Widget, SLOT(paste()),

QKeySequence::Paste);

\_ActionFind = \_MenuEdit->addAction(tr("Найти"),

\_Widget, SLOT(find()),

QKeySequence::Find);

\_ActionFindReplace = \_MenuEdit->addAction(tr("Найти и заменить"),

\_Widget, SLOT(findAndReplace()));

\_ActionSelectAll = \_MenuEdit->addAction(tr("Выделить все"),

\_Widget, SLOT(selectAll()),

QKeySequence::SelectAll);

\_MenuFormat = menuBar()->addMenu(tr("&Формат"));

\_ActionWordWrap = \_MenuFormat->addAction(tr("Перенос по словам"));

\_ActionWordWrap->setCheckable(true);

\_ActionWordWrap->setChecked(true);

\_ActionChangeFont = \_MenuFormat->addAction(tr("Выбор шрифта"),

\_Widget, SLOT(showChangeFont()));

\_MenuView = menuBar()->addMenu(tr("&Вид"));

\_ActionChangeBackground = \_MenuView->addAction(tr("Выбор цвета фона"),

\_Widget, SLOT(showChangeBackground()));

\_ActionChangeLineColor = \_MenuView->addAction(

tr("Выбор цвета текущей строки"),

\_Widget, SLOT(showChangeLineColor()));

\_ActionChangeNumeration = \_MenuView->addAction(

tr("Вкл/Выкл отображения нумерации строк"));

\_ActionChangeNumeration->setCheckable(true);

\_ActionChangeNumeration->setChecked(true);

\_ActionChangeToolBar = \_MenuView->addAction(

tr("Вкл/Выкл отображения панели инструментов"));

\_ActionChangeToolBar->setCheckable(true);

\_ActionChangeToolBar->setChecked(true);

\_ActionChangeState = \_MenuView->addAction(

tr("Вкл/Выкл отображения строки состояния"));

\_ActionChangeState->setCheckable(true);

\_ActionChangeState->setChecked(true);

\_ActionChangeHighlight = \_MenuView->addAction(

tr("Вкл/Выкл подсветки синтаксиса"));

\_ActionChangeHighlight->setCheckable(true);

\_ActionChangeHighlight->setChecked(true);

\_MenuChangeSyntax = \_MenuView->addMenu(tr("Выбор синтаксиса"));

\_C11 = \_MenuChangeSyntax->addAction(tr("Си 2011"),

\_Widget, SLOT(c11Syntax()));

\_C11->setCheckable(true);

\_C11->setChecked(true);

\_C18 = \_MenuChangeSyntax->addAction(tr("Си 2018"),

\_Widget, SLOT(c18Syntax()));

\_C18->setCheckable(true);

\_C18->setChecked(false);

\_Cpp14 = \_MenuChangeSyntax->addAction(tr("Си++ 2014"),

\_Widget, SLOT(cpp14Syntax()));

\_Cpp14->setCheckable(true);

\_Cpp14->setChecked(false);

\_Cpp17 = \_MenuChangeSyntax->addAction(tr("Си++ 2017"),

\_Widget, SLOT(cpp17Syntax()));

\_Cpp17->setCheckable(true);

\_Cpp17->setChecked(false);

\_Cpp20 = \_MenuChangeSyntax->addAction(tr("Си++ 2020"),

\_Widget, SLOT(cpp20Syntax()));

\_Cpp20->setCheckable(true);

\_Cpp20->setChecked(false);

\_MenuChangeStyle = \_MenuView->addMenu(

tr("Выбор/Редактирование стиля подсветки"));

\_ActionChangeStyle = \_MenuChangeStyle->addAction(tr("Изменить"),

this, SLOT(showChangeStyle()));

\_ActionLoadStyle = \_MenuChangeStyle->addAction(

tr("Загрузка стиля из файла"), this, SLOT(showLoadStyle()));

\_ActionDefaultStyle = \_MenuChangeStyle->addAction(tr("Default"));

\_ActionDefaultStyle->setCheckable(true);

\_ActionDefaultStyle->setChecked(true);

connect(\_MenuChangeStyle, SIGNAL(triggered(QAction\*)),

this, SLOT(chooseStyle(QAction\*)));

\_CurrStyleAction = \_ActionDefaultStyle;

\_Styles["Default"] = \_Widget->getStyle();

\_MenuReference = menuBar()->addMenu(tr("&Справка"));

\_ActionAbout = \_MenuReference->addAction(tr("О программе"),

\_Widget, SLOT(showAbout()));

connect(\_ActionWordWrap, &QAction::toggled,

\_Widget, &Widget::switchWordWrap);

connect(\_ActionChangeNumeration, &QAction::toggled,

\_Widget, &Widget::switchNumeration);

connect(\_ActionChangeToolBar, &QAction::toggled,

\_Widget, &Widget::switchToolBar);

connect(\_ActionChangeState, &QAction::toggled,

statusBar(), &QStatusBar::setVisible);

connect(\_ActionChangeHighlight, &QAction::toggled,

\_Widget, &Widget::switchHighlight);

\_ActionWordWrap->setChecked(true);

\_ActionChangeNumeration->setChecked(true);

\_ActionChangeToolBar->setChecked(true);

\_ActionChangeState->setChecked(true);

\_ActionChangeHighlight->setChecked(true);

\_Cursor = new QLabel("x: 0, y: 0");

\_Last = new QLabel("Последние сохранение: -; Последние изменение: -");

\_Amounts = new QLabel("Строки: 0, слова: 0, символы: 0, размер: 0");

statusBar()->addPermanentWidget(\_Cursor, 1);

statusBar()->addPermanentWidget(\_Last, 5);

statusBar()->addPermanentWidget(\_Amounts, 1);

QDir dir = QDir::currentPath();

QStringList filter;

filter << "\*.ini";

for(auto filename: dir.entryList(filter)) {

QSettings \*ss = new QSettings(filename,QSettings::IniFormat,this);

readStyle(ss);

QString styleName = QFileInfo(filename).baseName();

QAction \*act = \_MenuChangeStyle->addAction(styleName);

act->setCheckable(true);

\_Styles[act->text()] = \_CurrStyle;

}

\_CurrStyle = \_Styles["Default"];

}

void MainWindow::updateStatus() {

\_Cursor->setText("x: "+QString::number(\_Widget->getX())+

", y: " + QString::number(\_Widget->getY()));

if(\_Widget->\_Doc->isModified()){

\_Widget->\_LChange = QTime::currentTime();

\_Widget->\_LChangeDate = QDate::currentDate();

\_DayPassedC = false;

}

QString lst("Последние сохранение: ");

if(!(\_Widget->\_LSave.isNull())) {

if(\_DayPassedS)

lst += \_Widget->\_LSaveDate.toString();

else

lst += \_Widget->\_LSave.toString();

}

else

lst += "-";

lst += "; Последние изменение: ";

if(!(\_Widget->\_LChange.isNull())) {

if(\_DayPassedC)

lst += \_Widget->\_LChangeDate.toString();

else

lst += \_Widget->\_LChange.toString();

}

else

lst += "-";

\_Last ->setText(lst);

\_Amounts->setText("Строки: " + QString::number(\_Widget->lineCount()) +

"; Слова: " + QString::number(\_Widget->wordCount()) +

"; Символы: " + QString::number(\_Widget->symbCount()) +

"; Размер в килобайтах: " + QString::number(\_Widget->kbCount()));

QString title;

if(\_Widget->\_LChange>\_Widget->\_LSave) {

title = "\*";

}

title += \_Widget->\_File;

if(title.length()>32) {

title.chop(title.length()-32);

title+="...";

}

setWindowTitle(title);

}

void MainWindow::daySaved() {

\_DayPassedS = false;

updateStatus();

}

void MainWindow::selectC(QString c) {

\_C11->setChecked(false);

\_C18->setChecked(false);

\_Cpp14->setChecked(false);

\_Cpp17->setChecked(false);

\_Cpp20->setChecked(false);

if(c=="C11")

\_C11->setChecked(true);

else if(c=="C18")

\_C18->setChecked(true);

else if(c=="C++14")

\_Cpp14->setChecked(true);

else if(c=="C++17")

\_Cpp17->setChecked(true);

else

\_Cpp20->setChecked(true);

}

void MainWindow::passDay() {

\_DayPassedC = true;

\_DayPassedS = true;

updateStatus();

\_Timer->stop();

\_Timer->setSingleShot(false);

\_Timer->start(24\*3600\*1000);

}

void MainWindow::chooseStyle(QAction\* act) {

if(act != \_CurrStyleAction and act != \_ActionChangeStyle and

act != \_ActionLoadStyle) {

\_CurrStyleAction->setChecked(false);

act->setChecked(true);

\_Widget->setStyle(\_Styles[act->text()]);

\_CurrStyleAction = act;

}

}

void MainWindow::showChangeStyle() {

\_CurrStyle = \_Widget->getStyle();

QPalette p;

p.setColor(QPalette::Active, QPalette::Button,

\_CurrStyle.keywordFormat.foreground().color());

p.setColor(QPalette::Inactive,QPalette::Button,

\_CurrStyle.keywordFormat.foreground().color());

\_KeywordB->setPalette(p);

p.setColor(QPalette::Active, QPalette::Button,

\_CurrStyle.singleLineCommentFormat.foreground().color());

p.setColor(QPalette::Inactive, QPalette::Button,

\_CurrStyle.singleLineCommentFormat.foreground().color());

\_SLCommentB->setPalette(p);

p.setColor(QPalette::Active, QPalette::Button,

\_CurrStyle.multiLineCommentFormat.foreground().color());

p.setColor(QPalette::Inactive, QPalette::Button,

\_CurrStyle.multiLineCommentFormat.foreground().color());

\_MLCommentB->setPalette(p);

p.setColor(QPalette::Active, QPalette::Button,

\_CurrStyle.quotationFormat.foreground().color());

p.setColor(QPalette::Inactive, QPalette::Button,

\_CurrStyle.quotationFormat.foreground().color());

\_QuotationB->setPalette(p);

p.setColor(QPalette::Active, QPalette::Button,

\_CurrStyle.singleCharFormat.foreground().color());

p.setColor(QPalette::Inactive, QPalette::Button,

\_CurrStyle.singleCharFormat.foreground().color());

\_SCharB->setPalette(p);

p.setColor(QPalette::Active, QPalette::Button,

\_CurrStyle.functionFormat.foreground().color());

p.setColor(QPalette::Inactive, QPalette::Button,

\_CurrStyle.functionFormat.foreground().color());

\_FunctionB->setPalette(p);

p.setColor(QPalette::Active, QPalette::Button,

\_CurrStyle.directiveFormat.foreground().color());

p.setColor(QPalette::Inactive, QPalette::Button,

\_CurrStyle.directiveFormat.foreground().color());

\_DirectiveB->setPalette(p);

p.setColor(QPalette::Active, QPalette::Button,

\_CurrStyle.angleFormat.foreground().color());

p.setColor(QPalette::Inactive, QPalette::Button,

\_CurrStyle.angleFormat.foreground().color());

\_AngleB->setPalette(p);

\_StyleDialog->exec();

}

void MainWindow::showLoadStyle() {

QDir dir = QDir::currentPath();

QString fileName = QFileDialog::getOpenFileName(

\_StyleDialog, "Сохранить стиль",

dir.path(), "Initialization files (\*.ini)");

if(!fileName.isEmpty()){

QSettings \*styleSettings = new

QSettings(fileName,QSettings::IniFormat,this);

readStyle(styleSettings);

QString styleName = QFileInfo(fileName).baseName();

QAction \*act = \_MenuChangeStyle->addAction(styleName);

act->setCheckable(true);

act->setChecked(true);

\_CurrStyleAction->setChecked(false);

\_CurrStyleAction = act;

\_Styles[act->text()] = \_CurrStyle;

\_Widget->setStyle(\_Styles[act->text()]);

}

}

void MainWindow::readStyle(QSettings\* ss)

{

\_CurrStyle.directiveFormat.setForeground(

ss->value("style/directiveFormat").value<QColor>());

\_CurrStyle.functionFormat.setForeground(

ss->value("style/functionFormat").value<QColor>());

\_CurrStyle.keywordFormat.setForeground(

ss->value("style/keywordFormat").value<QColor>());

\_CurrStyle.multiLineCommentFormat.setForeground(

ss->value("style/multiLineCommentFormat").value<QColor>());

\_CurrStyle.quotationFormat.setForeground(

ss->value("style/quotationFormat").value<QColor>());

\_CurrStyle.singleCharFormat.setForeground(

ss->value("style/singleCharFormat").value<QColor>());

\_CurrStyle.singleLineCommentFormat.setForeground(

ss->value("style/singleLineCommentFormat").value<QColor>());

\_CurrStyle.angleFormat.setForeground(

ss->value("style/angleFormat").value<QColor>());

}

void MainWindow::createEditStyleDialog() {

\_StyleDialog = new QDialog(this);

QBoxLayout \*l = new QBoxLayout(QBoxLayout::TopToBottom);

QFormLayout \*tl = new QFormLayout();

\_KeywordB = new QPushButton(\_StyleDialog);

connect(\_KeywordB, SIGNAL(clicked()), this, SLOT(setStyleColor()));

\_SLCommentB = new QPushButton(\_StyleDialog);

connect(\_SLCommentB, SIGNAL(clicked()), this, SLOT(setStyleColor()));

\_MLCommentB = new QPushButton(\_StyleDialog);

connect(\_MLCommentB, SIGNAL(clicked()), this, SLOT(setStyleColor()));

\_QuotationB = new QPushButton(\_StyleDialog);

connect(\_QuotationB, SIGNAL(clicked()), this, SLOT(setStyleColor()));

\_SCharB = new QPushButton(\_StyleDialog);

connect(\_SCharB, SIGNAL(clicked()), this, SLOT(setStyleColor()));

\_FunctionB = new QPushButton(\_StyleDialog);

connect(\_FunctionB, SIGNAL(clicked()), this, SLOT(setStyleColor()));

\_DirectiveB = new QPushButton(\_StyleDialog);

connect(\_DirectiveB, SIGNAL(clicked()), this, SLOT(setStyleColor()));

\_AngleB = new QPushButton(\_StyleDialog);

connect(\_AngleB, SIGNAL(clicked()), this, SLOT(setStyleColor()));

QPushButton \*SaveB = new QPushButton(tr("Сохранить"), \_StyleDialog);

connect(SaveB, SIGNAL(clicked()), this, SLOT(saveStyle()));

tl->addRow("Keyword",\_KeywordB);

tl->addRow("Single comment",\_SLCommentB);

tl->addRow("Multiline comment",\_MLCommentB);

tl->addRow("Quotation",\_QuotationB);

tl->addRow("Char quotation",\_SCharB);

tl->addRow("Function",\_FunctionB);

tl->addRow("Directives",\_DirectiveB);

tl->addRow("Angle brackets",\_AngleB);

l->addLayout(tl);

l->addWidget(SaveB);

\_StyleDialog->setLayout(l);

}

void MainWindow::setStyleColor()

{

QColor c;

if(QObject::sender()==\_KeywordB)

c = \_CurrStyle.keywordFormat.foreground().color();

else if(sender()==\_SLCommentB)

c = \_CurrStyle.singleLineCommentFormat.foreground().color();

else if(sender()==\_MLCommentB)

c = \_CurrStyle.multiLineCommentFormat.foreground().color();

else if(sender()==\_QuotationB)

c = \_CurrStyle.quotationFormat.foreground().color();

else if(sender()==\_SCharB)

c = \_CurrStyle.singleCharFormat.foreground().color();

else if(sender()==\_FunctionB)

c = \_CurrStyle.functionFormat.foreground().color();

else if(sender()==\_DirectiveB)

c = \_CurrStyle.directiveFormat.foreground().color();

else if(sender()==\_AngleB)

c = \_CurrStyle.angleFormat.foreground().color();

else return;

QPalette p;

QColor nc = QColorDialog::getColor(c, \_StyleDialog);

if(nc.isValid())

{

p.setColor(QPalette::Active, QPalette::Button, nc);

p.setColor(QPalette::Inactive, QPalette::Button, nc);

if(QObject::sender()==\_KeywordB){

\_CurrStyle.keywordFormat.setForeground(nc);

\_KeywordB->setPalette(p);

}

else if(QObject::sender()==\_SLCommentB){

\_CurrStyle.singleLineCommentFormat.setForeground(nc);

\_SLCommentB->setPalette(p);

}

else if(QObject::sender()==\_MLCommentB){

\_CurrStyle.multiLineCommentFormat.setForeground(nc);

\_MLCommentB->setPalette(p);

}

else if(QObject::sender()==\_QuotationB){

\_CurrStyle.quotationFormat.setForeground(nc);

\_QuotationB->setPalette(p);

}

else if(QObject::sender()==\_SCharB){

\_CurrStyle.singleCharFormat.setForeground(nc);

\_SCharB->setPalette(p);

}

else if(QObject::sender()==\_FunctionB){

\_CurrStyle.functionFormat.setForeground(nc);

\_FunctionB->setPalette(p);

}

else if(QObject::sender()==\_DirectiveB){

\_CurrStyle.directiveFormat.setForeground(nc);

\_DirectiveB->setPalette(p);

}

else if(QObject::sender()==\_AngleB){

\_CurrStyle.angleFormat.setForeground(nc);

\_AngleB->setPalette(p);

}

}

}

void MainWindow::saveStyle()

{

QDir dir = QDir::currentPath();

QString fileName =

QFileDialog::getSaveFileName(\_StyleDialog, QString("Save Style"),

dir.path(), "Initialization files (\*.ini)");

if(!fileName.isEmpty()){

QString styleName = QFileInfo(fileName).baseName();

\_Widget->setStyle(\_CurrStyle);

if(fileName.right(4) != ".ini"){

fileName += ".ini";

}

QSettings \*ss = new QSettings(fileName,

QSettings::IniFormat,this);

writeStyle(ss);

ss->sync();

\_StyleDialog->close();

QAction \*act = \_MenuChangeStyle->addAction(styleName);

act->setCheckable(true);

\_Styles[act->text()] = \_CurrStyle;

\_CurrStyleAction = act;

\_CurrStyleAction->setChecked(true);

\_Widget->setStyle(\_CurrStyle);

}

}

void MainWindow::writeStyle(QSettings\* ss)

{

ss->setValue("style/keywordFormat",

\_CurrStyle.keywordFormat.foreground().color());

ss->setValue("style/singleLineCommentFormat",

\_CurrStyle.singleLineCommentFormat.foreground().color());

ss->setValue("style/multiLineCommentFormat",

\_CurrStyle.multiLineCommentFormat.foreground().color());

ss->setValue("style/quotationFormat",

\_CurrStyle.quotationFormat.foreground().color());

ss->setValue("style/singleCharFormat",

\_CurrStyle.singleCharFormat.foreground().color());

ss->setValue("style/functionFormat",

\_CurrStyle.functionFormat.foreground().color());

ss->setValue("style/directiveFormat",

\_CurrStyle.directiveFormat.foreground().color());

ss->setValue("style/angleFormat",

\_CurrStyle.angleFormat.foreground().color());

}

MenuBar::MenuBar(QWidget \*parent)

: QMenuBar(parent)

{

}

bool MenuBar::event(QEvent \*e)

{

// keep behavior of base class

bool ret = QMenuBar::event(e);

// check whether this is a help event

if (e->type() == QEvent::ToolTip) {

const QHelpEvent \*const pQHelpEvent = (const QHelpEvent\*)e;

const QAction \*pQAction = activeAction();

if (pQAction && !pQAction->toolTip().isEmpty()) {

QToolTip::showText(pQHelpEvent->globalPos(), pQAction->toolTip());

return ret;

}

}

QToolTip::hideText();

return ret;

}

Widget::Widget(MainWindow \*parent, int dh)

: QWidget(parent)

{

\_MainWindow = parent;

setMinimumSize(800,600-dh);

\_ToolBar = new MenuBar(this);

\_ToolBar->setDefaultUp(true);

\_ToolNew = \_ToolBar->addAction(tr("Новый"), this, SLOT(newFile()));

\_ToolNew->setIcon(QIcon(":images/new.png"));

\_ToolOpen = \_ToolBar->addAction(tr("Открыть"), this, SLOT(openFile()));

\_ToolOpen->setIcon(QIcon(":images/open.png"));

\_ToolSave = \_ToolBar->addAction(tr("Сохранить"), this, SLOT(saveFile()));

\_ToolSave->setIcon(QIcon(":images/save.png"));

\_ToolUndo = \_ToolBar->addAction(tr("Отменить"), this, SLOT(undo()));

\_ToolUndo->setIcon(QIcon(":images/undo.png"));

\_ToolRedo = \_ToolBar->addAction(tr("Повторить"), this, SLOT(redo()));

\_ToolRedo->setIcon(QIcon(":images/redo.png"));

\_ToolCopy = \_ToolBar->addAction(tr("Копировать"), this, SLOT(copy()));

\_ToolCopy->setIcon(QIcon(":images/copy.png"));

\_ToolCut = \_ToolBar->addAction(tr("Вырезать"), this, SLOT(cut()));

\_ToolCut->setIcon(QIcon(":images/cut.png"));

\_ToolPaste = \_ToolBar->addAction(tr("Вставить"), this, SLOT(paste()));

\_ToolPaste->setIcon(QIcon(":images/paste.png"));

\_ToolMenuFind = \_ToolBar->addMenu(tr("Найти / Найти и заменить"));

\_ToolMenuFind->setIcon(QIcon(":images/find.jpg"));

\_ToolFind = \_ToolMenuFind->addAction(tr("Найти"), this, SLOT(find()));

\_ToolReplace = \_ToolMenuFind->addAction(tr("Найти и заменить"),

this, SLOT(findAndReplace()));

\_TextField = new CodeEditor(this);

\_TextField->setGeometry(0,\_ToolBar->height()\*0.7,

800,600-dh/2-\_ToolBar->height()/0.7);

\_TextField->setContextMenuPolicy(Qt::NoContextMenu);

\_Doc = \_TextField->document();

\_Highlighter = new Highlighter(\_Doc);

connect(\_Doc, &QTextDocument::modificationChanged,

this, &QWidget::setWindowModified);

connect(\_Doc, &QTextDocument::contentsChanged,

parent, &MainWindow::updateStatus);

}

long Widget::getX() {

return \_TextField->textCursor().positionInBlock();

}

long Widget::getY() {

return \_TextField->textCursor().blockNumber() + 1;

}

long Widget::lineCount() {

return \_TextField->blockCount();

}

long Widget::wordCount() {

return \_TextField->toPlainText().split(" ",

Qt::SkipEmptyParts, Qt::CaseInsensitive).length() +

\_TextField->toPlainText().split("\n",

Qt::SkipEmptyParts, Qt::CaseInsensitive).length() - 1;

}

long Widget::symbCount() {

return \_TextField->toPlainText().length();

}

long Widget::kbCount() {

return QFileInfo(\_File).size()/1024;

}

Style Widget::getStyle() {

return \_Highlighter->getStyle();

}

void Widget::setStyle(Style s) {

\_Highlighter->setStyle(s);

}

void Widget::newFile() {

\_File.clear();

\_TextField->clear();

setWindowTitle(QString());

}

void Widget::openFile() {

QString fileName = QFileDialog::getOpenFileName(this, "Выберите файл");

QFile file(fileName);

\_File = fileName;

if (!file.open(QIODevice::ReadOnly | QFile::Text)) {

QMessageBox::warning(this, "Ошибка", "Не открыть файл: "

+ file.errorString());

return;

}

setWindowTitle(fileName);

QByteArray data = file.readAll();

QTextCodec \*codec = Qt::codecForHtml(data);

QString str = codec->toUnicode(data);

str = QString::fromLocal8Bit(data);

\_TextField->setPlainText(str);

file.close();

\_MainWindow->daySaved();

\_MainWindow->updateStatus();

}

void Widget::saveFile() {

if (\_File.isEmpty())

\_File = QFileDialog::getSaveFileName(this, "Сохранить");

QFile file(\_File);

if (!file.open(QIODevice::WriteOnly | QFile::Text)) {

QMessageBox::warning(this, "Ошибка", "Не сохранить файл: "

+ file.errorString());

return;

}

setWindowTitle(\_File);

QTextStream out(&file);

QString text = \_TextField->toPlainText();

out << text;

file.close();

\_LSave = QTime::currentTime();

\_LSaveDate = QDate::currentDate();

\_MainWindow->daySaved();

\_MainWindow->updateStatus();

}

void Widget::saveFileAs() {

QString \_File = QFileDialog::getSaveFileName(this, "Save as");

QFile file(\_File);

if (!file.open(QFile::WriteOnly | QFile::Text)) {

QMessageBox::warning(this, "Ошибка", "Не сохранить файл: "

+ file.errorString());

return;

}

setWindowTitle(\_File);

QTextStream out(&file);

QString text = \_TextField->toPlainText();

out << text;

file.close();

\_LSave = QTime::currentTime();

\_LSaveDate = QDate::currentDate();

\_MainWindow->daySaved();

\_MainWindow->updateStatus();

}

void Widget::undo() {

\_TextField->undo();

}

void Widget::redo() {

\_TextField->redo();

}

void Widget::copy() {

\_TextField->copy();

}

void Widget::cut() {

\_TextField->cut();

}

void Widget::paste() {

\_TextField->paste();

}

void Widget::find() {

QDialog \*FindDialog = new QDialog(this);

FindDialog->setModal(0);

FindDialog->setFixedSize(250, 100);

QFrame \*frame = new QFrame(this);

frame->setFrameStyle(0);

QLabel \*TextFind = new QLabel(tr("Найти: "), frame);

\_FindW = new QLineEdit(frame);

QPushButton \*NextB = new QPushButton(tr("Следующий"), frame);

QPushButton \*BackB = new QPushButton(tr("Предыдущий"), frame);

QGridLayout \*fl = new QGridLayout(frame);

fl->addWidget(TextFind, 0, 0);

fl->addWidget(\_FindW, 0, 1);

frame->setLayout(fl);

QGridLayout \*l = new QGridLayout(this);

l->addWidget(frame, 0, 0, 1, 0);

l->addWidget(NextB, 1, 0);

l->addWidget(BackB, 1, 1);

FindDialog->setLayout(l);

connect(NextB, &QPushButton::clicked, this, &Widget::findNext);

connect(BackB, &QPushButton::clicked, this, &Widget::findPrev);

FindDialog->show();

}

void Widget::findNext() {

\_TextField->find(\_FindW->text(), QTextDocument::FindWholeWords);

}

void Widget::findPrev() {

\_TextField->find(\_FindW->text(),

QTextDocument::FindWholeWords | QTextDocument::FindBackward);

}

void Widget::findAndReplace() {

QDialog \*FindDialog = new QDialog(this);

FindDialog->setModal(0);

FindDialog->setFixedSize(250, 125);

QFrame \*frame = new QFrame(this);

frame->setFrameStyle(0);

QLabel \*TextFind = new QLabel(tr("Найти: "), frame);

QLabel \*TextReplace = new QLabel(tr("Заменить: "), frame);

\_FindW = new QLineEdit(frame);

\_ReplaceW = new QLineEdit(frame);

QPushButton \*NextB = new QPushButton(tr("Следующее"), frame);

QPushButton \*BackB = new QPushButton(tr("Предыдущее"), frame);

QPushButton \*ReplB = new QPushButton(tr("Заменить"), frame);

QGridLayout \*fl = new QGridLayout(frame);

fl->addWidget(TextFind, 0, 0);

fl->addWidget(TextReplace, 1, 0);

fl->addWidget(\_FindW, 0, 1);

fl->addWidget(\_ReplaceW, 1, 1);

frame->setLayout(fl);

QGridLayout \*l = new QGridLayout(this);

l->addWidget(frame, 0, 0, 1, 0);

l->addWidget(NextB, 1, 0);

l->addWidget(BackB, 1, 1);

l->addWidget(ReplB, 1, 2);

FindDialog->setLayout(l);

connect(NextB, &QPushButton::clicked, this, &Widget::findNext);

connect(BackB, &QPushButton::clicked, this, &Widget::findPrev);

connect(ReplB, &QPushButton::clicked, this, &Widget::replace);

FindDialog->show();

}

void Widget::replace() {

if(\_ReplaceW!=0) {

\_TextField->textCursor().removeSelectedText();

\_TextField->textCursor().insertText(\_ReplaceW->text());

}

}

void Widget::selectAll() {

\_TextField->selectAll();

}

void Widget::switchWordWrap(bool arg) {

\_TextField->setLineWrapMode(

arg?(QPlainTextEdit::WidgetWidth):(QPlainTextEdit::NoWrap));

}

void Widget::showChangeFont() {

QFontDialog \*Dialog = new QFontDialog(this);

Dialog->exec();

\_TextField->setFont(Dialog->selectedFont());

}

void Widget::showChangeBackground() {

QColor NewColor = QColorDialog::getColor(

\_TextField->getBGColor(), this);

if(NewColor.isValid()) {

QPalette p;

p.setColor(QPalette::Active, QPalette::Base, NewColor);

p.setColor(QPalette::Inactive, QPalette::Base, NewColor);

\_TextField->setPalette(p);

}

}

void Widget::showChangeLineColor() {

QColor NewColor = QColorDialog::getColor(

\_TextField->getLColor(), this);

if(NewColor.isValid())

\_TextField->setLineColor(NewColor);

}

void Widget::switchNumeration(bool arg) {

\_TextField->flag = arg;

\_TextField->updateLineNumberAreaWidth(0);

}

void Widget::switchToolBar(bool arg) {

\_ToolBar->setVisible(arg);

}

void Widget::switchHighlight(bool arg) {

\_Highlighter->setVisibility(arg);

}

void Widget::c11Syntax() {

\_MainWindow->selectC("C11");

\_Highlighter->setStandart("C11/18");

}

void Widget::c18Syntax() {

\_MainWindow->selectC("C18");

\_Highlighter->setStandart("C11/18");

}

void Widget::cpp14Syntax() {

\_MainWindow->selectC("C++14");

\_Highlighter->setStandart("C++14");

}

void Widget::cpp17Syntax() {

\_MainWindow->selectC("C++17");

\_Highlighter->setStandart("C++17");

}

void Widget::cpp20Syntax() {

\_MainWindow->selectC("C++20");

\_Highlighter->setStandart("C++20");

}

void Widget::showAbout() {

QDialog\* Dialog = new QDialog(this);

Dialog->setFixedSize(500, 250);

QGridLayout \*l = new QGridLayout(Dialog);

Dialog->setWindowTitle("О программе");

QPixmap me( ":images/mememememe.jpg" );

me = me.scaled(350, 150, Qt::IgnoreAspectRatio);

QLabel \*pic = new QLabel(this);

pic->setPixmap(me);

QLabel \*dateOfBuild = new QLabel();

dateOfBuild->setText(tr(" Версия QT ") + QT\_VERSION\_STR +

"\n Тур Тимофей \n и мои 2 последние нервные клетки" +

"\n сделали это за 48 часов");

l->addWidget(pic, 0, 0, Qt::AlignCenter);

l->addWidget(dateOfBuild);

Dialog->exec();

}

Приложение И – codeeditor.cpp

#include "codeeditor.h"

#include <QPainter>

#include <QTextBlock>

CodeEditor::CodeEditor(QWidget \*parent)

: QPlainTextEdit(parent)

{

lineNumberArea = new LineNumberArea(this);

connect(this, &CodeEditor::blockCountChanged,

this, &CodeEditor::updateLineNumberAreaWidth);

connect(this, &CodeEditor::updateRequest,

this, &CodeEditor::updateLineNumberArea);

connect(this, &CodeEditor::cursorPositionChanged,

this, &CodeEditor::highlightCurrentLine);

updateLineNumberAreaWidth(0);

highlightCurrentLine();

}

void CodeEditor::lineNumberAreaPaintEvent(QPaintEvent \*event) {

QPainter painter(lineNumberArea);

painter.fillRect(event->rect(), Qt::lightGray);

QTextBlock block = firstVisibleBlock();

int blockNumber = block.blockNumber();

int top = qRound(blockBoundingGeometry(block).translated(

contentOffset()).top());

int bottom = top + qRound(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);

QFont f("Calibri");

painter.setFont(f);

painter.drawText(0, top, lineNumberArea->width(),

fontMetrics().height(),

Qt::AlignRight, number);

}

block = block.next();

top = bottom;

bottom = top + qRound(blockBoundingRect(block).height());

++blockNumber;

}

}

int CodeEditor::lineNumberAreaWidth() {

if(flag) {

int d = 1;

int mx = qMax(1, blockCount());

while (mx >= 10) {

mx /= 10;

++d;

}

return 3 + fontMetrics().horizontalAdvance(QLatin1Char('9')) \* d;

}

return 0;

}

void CodeEditor::resizeEvent(QResizeEvent \*e) {

QPlainTextEdit::resizeEvent(e);

QRect cr = contentsRect();

lineNumberArea->setGeometry(QRect(cr.left(), cr.top(),

lineNumberAreaWidth(), cr.height()));

}

void CodeEditor::updateLineNumberAreaWidth(int) {

setViewportMargins(lineNumberAreaWidth(), 0, 0, 0);

}

void CodeEditor::highlightCurrentLine() {

QList<QTextEdit::ExtraSelection> extraSelections;

if (!isReadOnly()) {

QTextEdit::ExtraSelection selection;

QColor lineColor = QColor(LColor).lighter(160);

selection.format.setBackground(lineColor);

selection.format.setProperty(QTextFormat::FullWidthSelection, true);

selection.cursor = textCursor();

selection.cursor.clearSelection();

extraSelections.append(selection);

}

setExtraSelections(extraSelections);

}

void CodeEditor::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);

}

QColor CodeEditor::getBGColor() const {

return BGColor;

}

QColor CodeEditor::getLColor() const {

return LColor;

}

bool CodeEditor::isVisibleLineNumberArea() const {

return lineNumberArea->isVisible();

}

void CodeEditor::setBackgroundColor(QColor newColor) {

BGColor = newColor;

highlightCurrentLine();

}

void CodeEditor::setLineColor(QColor newColor) {

LColor = newColor;

highlightCurrentLine();

}

LineNumberArea::LineNumberArea(CodeEditor \*editor)

: QWidget(editor)

{

codeEditor = editor;

}

QSize LineNumberArea::sizeHint() const {

return QSize(codeEditor->lineNumberAreaWidth(), 0);

}

void LineNumberArea::paintEvent(QPaintEvent \*event) {

codeEditor->lineNumberAreaPaintEvent(event);

}

# Приложение К – highlighter.cpp

#include "highlighter.h"

Highlighter::Highlighter(QTextDocument \*parent)

: QSyntaxHighlighter(parent), \_Visib(true)

{

\_DiPatt << "^#include" << "^#ifndef"<< "^#define" << "^#undef" << "^#if"

<< "^#ifndef" << "^#else" << "^#elif" << "^#endif" << "^#line"

<< "^#error" << "^#warning" << "^#pragma";

setStandart("C11/18");

setDefStyle(true);

}

void Highlighter::highlightBlock(const QString &text)

{

if(! \_Visib) return;

for(const HighlightingRule &rule: \_Rules) {

QRegularExpressionMatchIterator matchIterator =

rule.pattern.globalMatch(text);

while (matchIterator.hasNext()){

QRegularExpressionMatch match = matchIterator.next();

setFormat(match.capturedStart(), match.capturedLength(),

rule.format);

}

}

setCurrentBlockState(0);

int startIndex = 0;

if (previousBlockState() != 1)

startIndex = text.indexOf(comesspres);

while (startIndex >= 0){

QRegularExpressionMatch match = comespres.match(text, startIndex);

int endIndex = match.capturedStart();

int commentLength = 0;

if (endIndex == -1){

setCurrentBlockState(1);

commentLength = text.length() - startIndex;

}

else{

commentLength = endIndex - startIndex + match.capturedLength();

}

setFormat(startIndex, commentLength, \_Style.multiLineCommentFormat);

startIndex = text.indexOf(comesspres, startIndex + commentLength);

}

}

void Highlighter::setRules() {

\_Rules.clear();

HighlightingRule r;

for(const QString & pattern : \_KeyPatt) {

r.pattern = QRegularExpression(pattern);

r.format = \_Style.keywordFormat;

\_Rules.append(r);

}

r.pattern = QRegularExpression("\".\*\"");

r.format = \_Style.quotationFormat;

\_Rules.append(r);

r.pattern = QRegularExpression("\'.\*\'");

r.format = \_Style.singleCharFormat;

\_Rules.append(r);

r.pattern = QRegularExpression("\\b[A-Za-z0-9\_]+(?=\\()");

r.format = \_Style.functionFormat;

\_Rules.append(r);

r.pattern = QRegularExpression("//[^\n]\*");

r.format = \_Style.singleLineCommentFormat;

\_Rules.append(r);

comesspres = QRegularExpression("/\\\*");

comespres = QRegularExpression("\\\*/");

for(const QString& pattern : \_DiPatt) {

r.pattern = QRegularExpression(pattern);

r.format = \_Style.directiveFormat;

\_Rules.append(r);

}

r.pattern = QRegularExpression("<.\*>");

\_Rules.append(r);

rehighlight();

}

void Highlighter::setVisibility(bool f) {

\_Visib = f;

rehighlight();

}

void Highlighter::setStandart(QString arg) {

\_S = arg;

if (arg == "C11/18") {

\_KeyPatt.clear();

\_KeyPatt << "\\bauto\\b" << "\\bbreak\\b" << "\\bcase\\b"

<< "\\bchar\\b" << "\\bconst\\b" << "\\bcontinue\\b"

<< "\\bdefault\\b" << "\\bdo\\b" << "\\belse\\b"

<< "\\bdouble\\b" << "\\belse\\b" << "\\benum\\b"

<< "\\bextern\\b" << "\\bfloat\\b" << "\\bfor\\b"

<< "\\bgoto\\b" << "\\bif\\b" << "\\binline\\b" << "\\bint\\b"

<< "\\blong\\b" << "\\bregister\\b" << "\\brestrict\\b"

<< "\\breturn\\b" << "\\bshort\\b" << "\\bsigned\\b"

<< "\\bsizeof\\b" << "\\bstatic\\b" << "\\bstruct\\b"

<< "\\bswitch\\b" << "\\btypedef\\b" << "\\bunion\\b"

<< "\\bunsigned\\b" << "\\bvoid\\b" << "\\bvolatile\\b"

<< "\\bwhile\\b" << "\\balignas\\b" << "\\balignof\\b"

<< "\\batomic\_\\w+\\b" << "\\bbool\\b" << "\\bcomplex\\b"

<< "\\bimaginary\\b" << "\\bnoreturn\\b"

<< "\\bstatic\_assert\\b" << "\\bthread\_local\\b";

}

else if (arg == "C++14") {

\_KeyPatt.clear();

\_KeyPatt << "\\balignas\\b" << "\\balignof\\b" << "\\bchar16\_t\\b"

<< "\\bchar32\_t\\b" << "\\bconstexpr\\b" << "\\bdecltype\\b"

<< "\\bnoexcept\\b" << "\\bnullptr\\b"

<< "\\bstatic\_assert\\b" << "\\bthread\_local\\b"

<< "\\band\\b" << "\\band\_eq\\b" << "\\basm\\b"

<< "\\bauto\\b" << "\\bitand\\b" << "\\bitor\\b"

<< "\\bbool\\b" << "\\bbreak\\b" << "\\bcase\\b"

<< "\\bcatch\\b" << "\\bchar\\b" << "\\bclass\\b"

<< "\\bcompl\\b" << "\\bconst\\b" << "\\bconst\_cast\\b"

<< "\\bcontinue\\b" << "\\bdefault\\b" << "\\bdelete\\b"

<< "\\bdo\\b" << "\\bdouble\\b" << "\\bdynamic\_cast\\b"

<< "\\belse\\b" << "\\benum\\b" << "\\bexplicit\\b"

<< "\\bexport\\b" << "\\bextern\\b" << "\\bfalse\\b"

<< "\\bfloat\\b" << "\\bfor\\b" << "\\bfriend\\b"

<< "\\bgoto\\b" << "\\bif\\b" << "\\binline\\b"

<< "\\bint\\b" << "\\blong\\b" << "\\bmutable\\b"

<< "\\bnamespace\\b" << "\\bnew\\b" << "\\bnot\\b"

<< "\\bnot\_eq\\b" << "\\boperator\\b" << "\\bor\\b"

<< "\\bor\_eq\\b" << "\\bprivate\\b" << "\\bprotected\\b"

<< "\\bpublic\\b" << "\\bregister\\b"

<< "\\breinterpret\_cast\\b" << "\\breturn\\b"

<< "\\bshort\\b" << "\\bsigned\\b" << "\\bsizeof\\b"

<< "\\bstatic\\b" << "\\bstatic\_cast\\b" << "\\bstruct\\b"

<< "\\bswitch\\b" << "\\btemplate\\b" << "\\bthis\\b"

<< "\\bthrow\\b" << "\\btrue\\b" << "\\btry\\b"

<< "\\btypedef\\b" << "\\btypeid\\b" << "\\bunion\\b"

<< "\\bunsigned\\b" << "\\busing\\b" << "\\bvirtual\\b"

<< "\\bvoid\\b" << "\\bvolatile\\b" << "\\bwchar\_t\\b"

<< "\\bwhile\\b" << "\\bxor\\b" << "\\bxor\_eq\\b";

}

else if (arg == "C++17") {

\_KeyPatt.clear();

\_KeyPatt << "\\balignas\\b" << "\\balignof\\b" << "\\bchar16\_t\\b"

<< "\\bchar32\_t\\b" << "\\bconstexpr\\b" << "\\bdecltype\\b"

<< "\\bnoexcept\\b" << "\\bnullptr\\b" << "\\bstatic\_assert\\b"

<< "\\bthread\_local\\b" << "\\band\\b" << "\\band\_eq\\b"

<< "\\basm\\b" << "\\bauto\\b" << "\\bitand\\b" << "\\bitor\\b"

<< "\\bbool\\b" << "\\bbreak\\b" << "\\bcase\\b"

<< "\\bcatch\\b" << "\\bchar\\b" << "\\bclass\\b"

<< "\\bcompl\\b" << "\\bconst\\b" << "\\bconst\_cast\\b"

<< "\\bcontinue\\b" << "\\bdefault\\b" << "\\bdelete\\b"

<< "\\bdo\\b" << "\\bdouble\\b" << "\\bdynamic\_cast\\b"

<< "\\belse\\b" << "\\benum\\b" << "\\bexplicit\\b"

<< "\\bexport\\b" << "\\bextern\\b" << "\\bfalse\\b"

<< "\\bfloat\\b" << "\\bfor\\b" << "\\bfriend\\b"

<< "\\bgoto\\b" << "\\bif\\b" << "\\binline\\b" << "\\bint\\b"

<< "\\blong\\b" << "\\bmutable\\b" << "\\bnamespace\\b"

<< "\\bnew\\b" << "\\bnot\\b" << "\\bnot\_eq\\b"

<< "\\boperator\\b" << "\\bor\\b" << "\\bor\_eq\\b"

<< "\\bprivate\\b" << "\\bprotected\\b" << "\\bpublic\\b"

<< "\\bregister\\b" << "\\breinterpret\_cast\\b"

<< "\\breturn\\b" << "\\bshort\\b" << "\\bsigned\\b"

<< "\\bsizeof\\b" << "\\bstatic\\b" << "\\bstatic\_cast\\b"

<< "\\bstruct\\b" << "\\bswitch\\b" << "\\btemplate\\b"

<< "\\bthis\\b" << "\\bthrow\\b" << "\\btrue\\b" << "\\btry\\b"

<< "\\btypedef\\b" << "\\btypeid\\b" << "\\bunion\\b"

<< "\\bunsigned\\b" << "\\busing\\b" << "\\bvirtual\\b"

<< "\\bvoid\\b" << "\\bvolatile\\b" << "\\bwchar\_t\\b"

<< "\\bwhile\\b" << "\\bxor\\b" << "\\bxor\_eq\\b";

}

else if (arg == "C++20") {

\_KeyPatt.clear();

\_KeyPatt << "\\bchar8\_t\\b" << "\\bconcept\\b" << "\\bconsteval\\b"

<< "\\bconstinit\\b" << "\\bco\_await\\b" << "\\bco\_return\\b"

<< "\\bco\_yield\\b" << "\\brequires\\b" << "\\balignas\\b"

<< "\\balignof\\b" << "\\bchar16\_t\\b" << "\\bchar32\_t\\b"

<< "\\bconstexpr\\b" << "\\bdecltype\\b" << "\\bnoexcept\\b"

<< "\\bnullptr\\b" << "\\bstatic\_assert\\b"

<< "\\bthread\_local\\b" << "\\band\\b" << "\\band\_eq\\b"

<< "\\basm\\b" << "\\bauto\\b" << "\\bbitand\\b"

<< "\\bbitor\\b" << "\\bbool\\b" << "\\bbreak\\b"

<< "\\bcase\\b" << "\\bcatch\\b" << "\\bchar\\b"

<< "\\bclass\\b" << "\\bcompl\\b" << "\\bconst\\b"

<< "\\bconst\_cast\\b" << "\\bcontinue\\b" << "\\bdefault\\b"

<< "\\bdelete\\b" << "\\bdo\\b" << "\\bdouble\\b"

<< "\\bdynamic\_cast\\b" << "\\belse\\b" << "\\benum\\b"

<< "\\bexplicit\\b" << "\\bexport\\b" << "\\bextern\\b"

<< "\\bfalse\\b" << "\\bfloat\\b" << "\\bfor\\b"

<< "\\bfriend\\b" << "\\goto\\b" << "\\bif\\b"

<< "\\binline\\b" << "\\bint\\b" << "\\blong\\b"

<< "\\bmutable\\b" << "\\bnamespace\\b" << "\\bnew\\b"

<< "\\bnot\\b" << "\\bnot\_eq\\b" << "\\boperator\\b"

<< "\\bor\\b" << "\\bor\_eq\\b" << "\\bprivate\\b"

<< "\\bprotected\\b" << "\\bpublic\\b"

<< "\\bregister\\b" << "\\breinterpret\_cast\\b"

<< "\\breturn\\b" << "\\bshort\\b" << "\\bsigned\\b"

<< "\\bsizeof\\b" << "\\bstatic\\b" << "\\bstatic\_cast\\b"

<< "\\bstruct\\b" << "\\bswitch\\b" << "\\btemplate\\b"

<< "\\bthis\\b" << "\\bthrow\\b" << "\\btrue\\b" << "\\btry\\b"

<< "\\btypedef\\b" << "\\btypeid\\b" << "\\bunion\\b"

<< "\\bunsigned\\b" << "\\busing\\b" << "\\bvirtual\\b"

<< "\\bvoid\\b" << "\\bvolatile\\b" << "\\bwchar\_t\\b"

<< "\\bwhile\\b" << "\\bxor\\b" << "\\bxor\_eq\\b";

}

else return;

setRules();

}

bool Highlighter::getVisible() const {

return \_Visib;

}

QString Highlighter::getStandart() const {

return \_S;

}

Style Highlighter::getStyle() const {

return \_Style;

}

void Highlighter::setDefStyle(bool change) {

if(! change) return;

\_Style.functionFormat.setForeground(Qt::darkCyan);

\_Style.quotationFormat.setForeground(Qt::darkGreen);

\_Style.keywordFormat.setForeground(Qt::darkRed);

\_Style.keywordFormat.setFontWeight(QFont::Bold);

\_Style.singleLineCommentFormat.setForeground(Qt::darkGray);

\_Style.multiLineCommentFormat.setForeground(Qt::darkGray);

\_Style.directiveFormat.setForeground(Qt::darkMagenta);

\_Style.directiveFormat.setFontWeight(QFont::Bold);

\_Style.singleCharFormat.setForeground(Qt::darkGreen);

setRules();

}

void Highlighter::setStyle(Style style) {

\_Style = style;

\_Style.directiveFormat.setFontWeight(QFont::Bold);

\_Style.keywordFormat.setFontWeight(QFont::Bold);

setRules();

}