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 СКБ-201	______________________ подпись, дата	Г.П. Кашкин
Работу проверил	______________________ подпись, дата	М.Ю. Монина

Москва 2021

Содержание

1 Алгоритм решения задачи

1.1 Текстовый редактор

На базе QPlainTextEdit был создан кастомный класс CodeEditor с базовой логикой текстового редактора и полями под указатели на класс подсветки синтаксиса и классы поиска/замены.

1.2 Подсветка синтаксиса

Реализован класс подсветки синтаксиса как наследник QSyntaxHighlighter включающий в себя интерфейс для конфигурации, поля типа подсветки (через enum class с акомпонимирующим массивом строк-названий для каждого значения енама). Правила подсветки хранятся в виде строк-литералов и передаются в QStringList паттернов.

1.3 Поиск и замена

Модуль позволяет делать гибкий поиск и замену строк в тексте через всплывающее модальное окно.

1.4 Окно приложения

Окно приложения реализовано как отдельный класс, создает требуемый интерфейс и соединяет необходимые пары слотов и сигналов.

2 Выполнение задания

2.1 Задание 1

2.2 Текстовый редактор

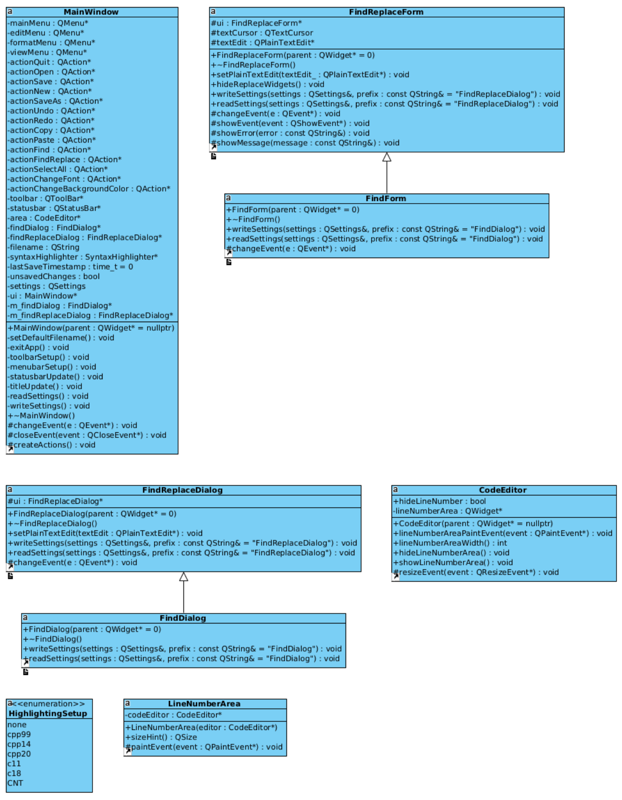
MainWindow
-mainMenu : QMenu*
-editMenu : QMenu*
-formatMenu : QMenu*
-viewMenu : QMenu*
-actionQuit : QAction*
-actionOpen : QAction*
-actionSave : QAction*
-actionNew : QAction*
-actionSaveAs : QAction*
-actionUndo : QAction*
-actionRedo : QAction*
-actionCopy : QAction*
-actionPaste : QAction*
-actionFind : QAction*
-actionFindReplace : QAction*
-actionSelectAll : QAction*
-actionChangeFont : QAction*
-actionChangeBackgroundColor : QAction*
-toolbar : QToolBar*
-statusbar : QStatusBar*
-area : CodeEditor*
-findDialog : FindDialog*
-findReplaceDialog : FindReplaceDialog*
-filename : QString
-syntaxHighlighter : SyntaxHighlighter*
-lastSaveTimestamp : time_t = 0
-unsavedChanges : bool
-settings : QSettings
-ui : MainWindow*
-m_findDialog : FindDialog*
-m_findReplaceDialog : FindReplaceDialog*
+MainWindow(parent : QWidget* = nullptr)
-setDefaultFilename() : void
-exitApp() : void
-toolbarSetup() : void
-menubarSetup() : void
-statusbarUpdate() : void
-titleUpdate() : void
-readSettings() : void
-writeSettings() : void
+~MainWindow()
#changeEvent(e : QEvent*) : void
#closeEvent(event : QCloseEvent*) : void
#createActions() : void
FindReplaceForm
#ui : FindReplaceForm*
#textCursor : QTextCursor
#textEdit : QPlainTextEdit*
+FindReplaceForm(parent : QWidget* = 0)
+~FindReplaceForm()
+setPlainTextEdit(textEdit_ : QPlainTextEdit*) : void
+hideReplaceWidgets() : void
+writeSettings(settings : QSettings&, prefix : const QString& = "FindReplaceDialog") : void
+readSettings(settings : QSettings&, prefix : const QString& = "FindReplaceDialog") : void
#changeEvent(e : QEvent*) : void
#showEvent(event : QShowEvent*) : void
#showError(error : const QString&) : void
#showMessage(message : const QString&) : void
FindForm
+FindForm(parent : QWidget* = 0)
+~FindForm()
+writeSettings(settings : QSettings&, prefix : const QString& = "FindDialog") : void
+readSettings(settings : QSettings&, prefix : const QString& = "FindDialog") : void
#changeEvent(e : QEvent*) : void
FindReplaceDialog
#ui : FindReplaceDialog*
+FindReplaceDialog(parent : QWidget* = 0)
+~FindReplaceDialog()
+setPlainTextEdit(textEdit : QPlainTextEdit*) : void
+writeSettings(settings : QSettings&, prefix : const QString& = "FindReplaceDialog") : void
+readSettings(settings : QSettings&, prefix : const QString& = "FindReplaceDialog") : void
#changeEvent(e : QEvent*) : void
CodeEditor
+hideLineNumber : bool
-lineNumberArea : QWidget*
+CodeEditor(parent : QWidget* = nullptr)
+lineNumberAreaPaintEvent(event : QPaintEvent*) : void
+lineNumberAreaWidth() : int
+hideLineNumberArea() : void
+showLineNumberArea() : void
#resizeEvent(event : QResizeEvent*) : void
FindDialog
+FindDialog(parent : QWidget* = 0)
+~FindDialog()
+writeSettings(settings : QSettings&, prefix : const QString& = "FindDialog") : void
+readSettings(settings : QSettings&, prefix : const QString& = "FindDialog") : void
<<enumeration>>
HighlightingSetup
none
cpp99
cpp14
cpp20
c11
c18
CNT
LineNumberArea
-codeEditor : CodeEditor*
+LineNumberArea(editor : CodeEditor*)
+sizeHint() : QSize
#paintEvent(event : QPaintEvent*) : void

Рис. 1. UML 2.0 diagram for gCodeEditor

Класс текстового редактора был пронаследован от QPlainTextEdit. Из добавочной логики можно отметить отключаемую нумерацию строк, которая реализована как отдельная зона левее зоны текстового редактора, ширина этой зоны вычисляется динамически в зависимости от количества строк.

2.3 Подсветка синтаксиса

Класс подсветки синтаксиса был пронаследован от QSyntaxHighlighter, был добавлен интерфейс конфигурации, позволяющий изменять стиль подсветки и выбирать конкретную версию языка.

2.4 Поиск и замена

Для решения данной задачи был найден модуль диалога поиска замены, который был в свою очередь доработан для совместимости с QPlainTextEdit (https://github.com/Lord-KA/QtFindReplaceDialogGitHub). Модуль подтягивается прямо с github по тегу при помощи CMake и собирается в статическую библиотеку, которая в свою очередь подключается к основному проекту.

2.5 Окно приложения

Окно приложения реализовано как отдельный класс, создает требуемый интерфейс, создает и конфигурирует вспомогательные объекты такие как SyntaxHighlighter, CodeEditor и FindReplaceDialog.

3 Получение исполняемых модулей

Для всего проекта использовалась система сборки `cmake`. В конфигурации системы сборки прописаны пять режимов компиляции: basic, sanitizer, release, debug и coverage с разными наборами флагов компиляции (использование посредством флага `-DCMAKE_BUILD_TYPE=*`), в каждом из них прописано использование требуемого стандарта `c++20`, а для проверки на покрытие тестов специфицируется компилятор `clang++`, из-за отсутствия универсального набора флагов у основных компиляторов. Помимо этого `cmake` автоматически скачивает с github и подключает библиотеку с реализацией FindReplaceDialog.

Приложение А

subsectionФайл CMakeLists.txt

```
cmake_minimum_required(VERSION 3.1.0)

project(Lab_5)

set(CMAKE_AUTOMOC ON)
set(CMAKE_AUTORCC ON)
set(CMAKE_AUTOUIC ON)

if(CMAKE_VERSION VERSION_LESS "3.7.0")
    set(CMAKE_INCLUDE_CURRENT_DIR ON)
endif()

set(CMAKE_CXX_FLAGS "${CMAKE_CXX_FLAGS} -std=c++20 -g -D EXTRA_VERBOSE"
   CACHE STRING "Comment" FORCE)
set(CMAKE_CXX_FLAGS_SANITIZER "${CMAKE_CXX_FLAGS} -Wpedantic -Wall -Wextra
   -Wformat=2 -fsanitize=address,undefined -g" CACHE STRING "Comment" FORCE
   )

find_package(Qt5 COMPONENTS Widgets REQUIRED)

include(FetchContent)

FetchContent_Declare(
  frdialog
  GIT_REPOSITORY https://github.com/lord-ka/QtFindReplaceDialog.git
)

FetchContent_GetProperties(frdialog)
if(NOT frdialog_POPULATED)
  FetchContent_Populate(frdialog)
  add_subdirectory(${frdialog_SOURCE_DIR}/dialogs)
endif()


add_executable( run
    main.cpp
    MainWindow.cpp
    TextEditor.cpp
    MainWindow.hpp
    TextEditor.hpp
    SyntaxHighlighter.hpp
    SyntaxHighlighter.cpp
)

target_link_libraries(run Qt5::Widgets)
target_link_libraries(run QtFindReplaceDialog)
```

Приложение Б

subsectionФайл `main.cpp`

```
#include "MainWindow.hpp"
#include <QtWidgets/QApplication>

int main(int argc, char *argv[])
{
    QApplication app(argc, argv);
    MainWindow window;
    window.show();
    return app.exec();
}
```

Приложение В

В.1 Файл MainWindow.hpp

```
#pragma once

#include <QVector>
#include <QPushButton>
#include <QMenuBar>
#include <QCoreApplication>
#include <QContextMenuEvent>
#include <QToolBar>
#include <QPainter>
#include <QMainWindow>
#include <QString>
#include <QFileDialog>
#include <QFile>
#include <QMessageBox>
#include <QHBoxLayout>
#include <QComboBox>
#include <QFont>
#include <QSpinBox>
#include <QCheckBox>
#include <QWidgetAction>
#include <QColorDialog>
#include <QStatusBar>

#include <iostream>
#include <string>
#include <fstream>
#include <chrono>
#include <ctime>

#include "TextEditor.hpp"
#include "SyntaxHighlighter.hpp"

#include "finddialog.h"
#include "findreplacedialog.h"

static const char DEFAULT_FILENAME[] = "untitled";
static const char DEFAULT_FILETYPE[] = "txt";

static const QFont::StyleHint DEFAULT_FONT = QFont::Times;

class MainWindow : public QMainWindow
{
  Q_OBJECT

public:
  MainWindow(QWidget *parent = nullptr);

private:
    void setDefaultFilename();
    void exitApp();

    void toolbarSetup();
    void menubarSetup();

    void statusbarUpdate();
    void titleUpdate();

public slots:
    void quit();
    void openFile();
    void newFile();
    void saveFile();
    void saveAsFile();

    void changeFont();

    void changeBackgroundColor();

private:
  QMenu *mainMenu, *editMenu, *formatMenu, *viewMenu;
    QAction *actionQuit, *actionOpen, *actionSave, *actionNew, *
   actionSaveAs;
    QAction *actionUndo, *actionRedo, *actionCopy, *actionPaste, *
   actionFind, *actionFindReplace, *actionSelectAll;
    QAction *actionChangeFont;
    QAction *actionChangeBackgroundColor;

    QToolBar   *toolbar;
    QStatusBar *statusbar;
    CodeEditor *area;
    FindDialog        *findDialog;
    FindReplaceDialog *findReplaceDialog;
    QString filename;

    SyntaxHighlighter *syntaxHighlighter;

    std::time_t lastSaveTimestamp = 0;

    bool unsavedChanges;
};
```

B.2 Файл MainWindow.cpp

```
#include "MainWindow.hpp"

MainWindow::MainWindow(QWidget* parent)
  : QMainWindow(parent)
{
    toolbarSetup();

    menubarSetup();

    area = new CodeEditor();

    statusbar = new QStatusBar(this);
    setStatusBar(statusbar);

    findDialog = new FindDialog(this);
    findDialog->setModal(false);
    findDialog->setPlainTextEdit(area);

    findReplaceDialog = new FindReplaceDialog(this);
    findReplaceDialog->setModal(false);
    findReplaceDialog->setPlainTextEdit(area);

    setCentralWidget(area);
    setMinimumSize(1280, 720);

    setDefaultFilename();

    syntaxHighlighter = new SyntaxHighlighter(area->document(), HighlightingSetup::none, "tomorrow");

    unsavedChanges = false;
    titleUpdate();
    statusbarUpdate();
    connect(area, &QPlainTextEdit::textChanged, this, [this](){statusbarUpdate(); unsavedChanges = true; titleUpdate();});
}

void MainWindow::toolbarSetup()
{
    toolbar = addToolBar("ToolBar");
    toolbar->addAction("New",   this, SLOT(newFile()));
    toolbar->addAction("Open",  this, SLOT(openFile()));
    toolbar->addAction("Undo",  this, [&, this](){area->undo();});
    toolbar->addAction("Redo",  this, [&, this](){area->redo();});
    toolbar->addAction("Copy",  this, [&, this](){area->copy();});
    toolbar->addAction("Paste", this, [&, this](){area->paste();});
    toolbar->addAction("Find",               this, [&, this](){findDialog->show();});
    toolbar->addAction("Find and Replace",   this, [&, this](){findReplaceDialog->show();});
}

void MainWindow::menubarSetup()
{
    /* Main Menu setup */
  mainMenu = menuBar()->addMenu("File");
  actionQuit   = mainMenu->addAction("Quit",    this, SLOT(quit()));
  actionOpen   = mainMenu->addAction("Open",    this, SLOT(openFile()));
  actionNew    = mainMenu->addAction("New",     this, SLOT(newFile()));
  actionSave   = mainMenu->addAction("Save",    this, SLOT(saveFile()));
  actionSaveAs = mainMenu->addAction("Save as", this, SLOT(saveAsFile()));

    /* Edit Menu setup */
    editMenu = menuBar()->addMenu("Edit");
    actionUndo  = editMenu->addAction("Undo",  this, [&, this](){area->undo();});
    actionRedo  = editMenu->addAction("Redo",  this, [&, this](){area->redo();});
    actionCopy  = editMenu->addAction("Copy",  this, [&, this](){area->copy();});
    actionPaste = editMenu->addAction("Paste", this, [&, this](){area->paste();});
    actionFind        = editMenu->addAction("Find",              this, [&, this](){findDialog->show();});
    actionFindReplace = editMenu->addAction("Find and Replace",  this, [&, this](){findReplaceDialog->show();});
    actionSelectAll   = editMenu->addAction("Select All",        this, [&, this](){area->selectAll();});

    /* Format Menu setup */
    formatMenu = menuBar()->addMenu("Format");
    actionChangeFont = formatMenu->addAction("Change font", this, SLOT(changeFont()));
    QWidgetAction *actionWrapperBox = new QWidgetAction(formatMenu);
    QCheckBox *wrapperBox = new QCheckBox(formatMenu);
    wrapperBox->setText("Enable wrapping");
    actionWrapperBox->setDefaultWidget(wrapperBox);
    connect(wrapperBox, &QCheckBox::stateChanged, this, [this, wrapperBox]()
            {
                area->setLineWrapMode((QPlainTextEdit::LineWrapMode)!wrapperBox->checkState());
            });
    formatMenu->addAction(actionWrapperBox);

    /* View Menu setup */
    viewMenu = menuBar()->addMenu("View");
    actionChangeBackgroundColor = viewMenu->addAction("Change background color", this, SLOT(changeBackgroundColor()));
    QWidgetAction *actionHideLineNumBox = new QWidgetAction(formatMenu);
    QCheckBox *hideLineNumBox = new QCheckBox(formatMenu);
    hideLineNumBox->setText("Hide line numbers");
    actionHideLineNumBox->setDefaultWidget(hideLineNumBox);
    connect(hideLineNumBox, &QCheckBox::stateChanged, this, [this, hideLineNumBox]()
            {
                if (hideLineNumBox->checkState())
                    area->hideLineNumberArea();
                else
                    area->showLineNumberArea();
                area->updateLineNumberAreaWidth();
            });
    viewMenu->addAction(actionHideLineNumBox);
    viewMenu->addAction(toolbar->toggleViewAction());
    /* Highlighting setup */            //TODO
    QActionGroup *highlightingGroup = new QActionGroup(viewMenu);
    for (int i = 0; i < static_cast<int>(HighlightingSetup::CNT); ++i) {
        QAction *action = new QAction(HighlightingSetups[i], highlightingGroup);
        connect(action, &QAction::triggered, this, [this, i](){this->syntaxHighlighter->setSyntax(static_cast<HighlightingSetup>(i));});
        action = highlightingGroup->addAction(viewMenu->addAction(HighlightingSetups[i], this, [this, i](){syntaxHighlighter->setSyntax(static_cast<HighlightingSetup>(i));}));
        highlightingGroup->addAction(action);
        if (i == 0)
            action->setChecked(true);
    }
    highlightingGroup->setVisible(true);
}

void MainWindow::statusbarUpdate()
{
    QTextCursor cursor = area->textCursor();
    size_t line = cursor.blockNumber() + 1;
    size_t pos  = cursor.positionInBlock() + 1;
    cursor.movePosition(QTextCursor::End);
    size_t lineCount  = cursor.blockNumber() + 1;
    size_t charsCount = cursor.position() + 1;
    size_t wordCount  = area->toPlainText().split(QRegExp("(\\s|\\n|\\r)+"),
                                                           QString::SkipEmptyParts).count();


    #ifdef EXTRA_VERBOSE
        fprintf(stderr, "line = %lu, pos = %lu\n", line, pos);
    #endif

    std::tm *now = std::localtime(&lastSaveTimestamp);
    QString message =          QString::number(line) +          ":" + QString::number(pos) +
                       " | "   + QString::number(now->tm_hour) +  ":" + QString::number(now->tm_min) +   ":" + QString::number(now->tm_sec) +
                       " | l:" + QString::number(lineCount) + " | w:" + QString::number(wordCount) + " | c:" + QString::number(charsCount) +
                       " |Kb:" + QString::number(charsCount / 1024);
    statusbar->showMessage(message);
}

void MainWindow::titleUpdate()
{
    QString title = filename;
    if (unsavedChanges)
        title += "*";
    setWindowTitle(title);
}

void MainWindow::changeBackgroundColor()
```

```
{
    QColorDialog *dialog = new QColorDialog(QColor("white"), this);
    dialog->exec();

    QPalette p = area->palette();
    p.setColor(QPalette::Active, QPalette::Base, dialog->selectedColor());
    area->setPalette(p);
    area->setBackgroundVisible(false);
}

void MainWindow::changeFont()
{
    #ifdef EXTRA_VERBOSE
        std::cerr << "Changing font to ";
    #endif
    QFont currentFont = area->document()->defaultFont();
    int fontSize = currentFont.pointSize();

    QDialog *askFont = new QDialog(this);
    askFont->setLayout(new QHBoxLayout());

    QComboBox *fontBox = new QComboBox(askFont);
    fontBox->addItem("Helvetica");
    fontBox->addItem("Times");
    fontBox->addItem("Courier");
    fontBox->addItem("OldEnglish");
    fontBox->addItem("System");
    fontBox->addItem("Any");

    fontBox->setCurrentIndex((int)currentFont.styleHint());
    QPushButton *buttonOk     = new QPushButton("Ok");
    QPushButton *buttonCancel = new QPushButton("Cancel");

    QSpinBox *sizeBox = new QSpinBox(askFont);
    sizeBox->setMinimum(1);
    sizeBox->setValue(currentFont.pointSize());

    askFont->layout()->addWidget(fontBox);
    askFont->layout()->addWidget(sizeBox);
    askFont->layout()->addWidget(buttonOk);
    askFont->layout()->addWidget(buttonCancel);
    connect(buttonOk, &QPushButton::clicked, this, [&currentFont, &fontBox,
    &sizeBox, askFont]()
            {
                currentFont.setStyleHint((QFont::StyleHint)fontBox->
   currentIndex());
                askFont->accept();
                currentFont.setPointSize(sizeBox->value());
            });
    connect(buttonCancel, &QPushButton::clicked, this, [askFont]()
            {
                askFont->reject();
            });

    askFont->exec();

    area->document()->setDefaultFont(currentFont);

    #ifdef EXTRA_VERBOSE
        std::cerr << currentFont.styleHint() << "!\n";
    #endif
}

void MainWindow::exitApp()
{
    #ifdef EXTRA_VERBOSE
        std::cerr << "Exiting the application!\n";
    #endif

    exit(0);
}

void MainWindow::quit()
{
    #ifdef EXTRA_VERBOSE
        std::cerr << "Quit Dialog!\n";
    #endif

    QMessageBox::StandardButton reply = QMessageBox::question(this, "Exit", "Close without saving?",
                                                                       QMessageBox::Save   |
                                                                       QMessageBox::Cancel |
                                                                       QMessageBox::Close    );
    if (reply == QMessageBox::Save) {
        saveFile();
        exitApp();
    } else if (reply == QMessageBox::Close) {
        exitApp();
    }
}

void MainWindow::setDefaultFilename()
{
    int num = 1;
    std::ifstream file;
    do {
        file.close();
        filename = DEFAULT_FILENAME + QString::number(num) + "." + DEFAULT_FILETYPE;
        file.open(filename.toStdString(), std::ifstream::in);
        ++num;
    } while (file.good());
}
```

```

void MainWindow::saveFile()
{
    QString text = area->toPlainText();

    if (filename.isEmpty()) {
        setDefaultFilename();
    }
    lastSaveTimestamp = std::time(0);
    statusbarUpdate();
    unsavedChanges = false;
    titleUpdate();

    #ifdef EXTRA_VERBOSE
        std::cerr << "Saving File to " << filename.toStdString() << "!\n";
    #endif

    QFile fout(filename);
    fout.open(QIODevice::ReadWrite | QIODevice::Text);
    fout.write(text.toUtf8());
    fout.write("\n");
    fout.close();
}

void MainWindow::saveAsFile()
{
    filename = QFileDialog::getSaveFileName(this, "Save file as", ".");
    lastSaveTimestamp = std::time(0);
    statusbarUpdate();
    unsavedChanges = false;
    titleUpdate();

    #ifdef EXTRA_VERBOSE
        std::cerr << "Saving File As to " << filename.toStdString() << "!\n
   ";
    #endif

    saveFile();
}

void MainWindow::openFile()
{
    filename = QFileDialog::getOpenFileName(this, "Open", ".");
    lastSaveTimestamp = std::time(0);
    statusbarUpdate();
    unsavedChanges = false;
    titleUpdate();

    #ifdef EXTRA_VERBOSE
        std::cerr << "Opening File " << filename.toStdString() << "!\n";
    #endif

    QFile file(filename);
    if (file.open(QIODevice::ReadWrite | QIODevice::Text)) {
        area->setPlainText(file.readAll());
        file.close();
    }
}

void MainWindow::newFile()
{

    setDefaultFilename();
    lastSaveTimestamp = std::time(0);
    statusbarUpdate();
    unsavedChanges = false;
    titleUpdate();

    #ifdef EXTRA_VERBOSE
        std::cerr << "Creating new File " << filename.toStdString() << " !\n";
    #endif
}
```

Приложение Г

Г.1 Файл SyntaxHighlighter.hpp

```
#pragma once

#include <QSyntaxHighlighter>
#include <QString>
#include <QTextDocument>
#include <QRegularExpression>

#include <iostream>

static const size_t MAX_FILETYPE_LEN = 100;
static const char SUPPORTED_THEMES[][MAX_FILETYPE_LEN] = {"monokai", "
   tomorrow", "tomorrowNight", "solarized"};

enum class HighlightingSetup {
    none = 0,
    cpp99,
    cpp14,
    cpp20,
    c11,
    c18,
    CNT,
};

static const char HighlightingSetups[][MAX_FILETYPE_LEN] = {
        "none",
        "C++99",
        "C++14",
        "C++20",
        "C11",
        "C18",
    };

class SyntaxHighlighter : QSyntaxHighlighter {
public:
    SyntaxHighlighter(QTextDocument *parent, HighlightingSetup setup,
   QString theme);

    void setupSyntaxHighlighter(HighlightingSetup setup, QString theme);

    void setSyntax(HighlightingSetup setup);
    void setTheme (HighlightingSetup setup, QString theme);

    void setLangRules();

    void setupKeywordPatterns();

    void setColorValues(QString theme);

    void highlightBlock(const QString &text);


private:
    HighlightingSetup setup;
    QString theme;

    QRegularExpression commentStartExpression, commentEndExpression;
    bool commonTextColorIsWhite = 1;

    QStringList difKeywordPatterns;

    struct HighlightingRule
    {
        QRegularExpression pattern;
        QTextCharFormat format;
    };

    QVector<HighlightingRule> highlightingRules;

    QTextCharFormat keywordFormat;
    QTextCharFormat classFormat;
    QTextCharFormat singleLineCommentFormat;
    QTextCharFormat multiLineCommentFormat;
    QTextCharFormat quotationFormat;
    QTextCharFormat functionFormat;
    QTextCharFormat keywordPatterns;
    QTextCharFormat keywordPatterns_C;
    QTextCharFormat keywordPatterns_Python;
    QTextCharFormat headerFileFormat;
    QTextCharFormat numberFormat;
    QTextCharFormat formatStringFormat;
    QTextCharFormat operatorFormat;
    QTextCharFormat phpVarFormat;
    QTextCharFormat rubyVarFormat;
    QTextCharFormat tagFormat;
    QTextCharFormat valueFormat;
    QTextCharFormat attributeFormat;
    QTextCharFormat idFormat;

    QColor keywordColor;
    QColor keyword2Color;
    QColor functionsColor;
    QColor valueColor;
    QColor numColor;
    QColor operatorColor;
    QColor formatStringColor;
    QColor commentColor;
    QColor varColor;
    QColor tagColor;
    QColor htmlAttributesColor;
    QColor cssClassesIDsColor;
    QColor cssAttributesColor;
};
```

Г.2 Файл SyntaxHighlighter.cpp

```cpp
#include "SyntaxHighlighter.hpp"

SyntaxHighlighter::SyntaxHighlighter(QTextDocument *parent,
    HighlightingSetup setup, QString theme)
    : QSyntaxHighlighter(parent), setup(setup)
{
    commentStartExpression = QRegularExpression("");
    commentEndExpression = QRegularExpression("");
    setupSyntaxHighlighter(setup, theme);
}

void SyntaxHighlighter::setupSyntaxHighlighter(HighlightingSetup setup,
    QString theme)
{
    setColorValues(theme);

    setSyntax(setup);
}

void SyntaxHighlighter::setSyntax(HighlightingSetup newSetup)
{
    setup = newSetup;
    setupKeywordPatterns();
    setLangRules();
}

void SyntaxHighlighter::setTheme(HighlightingSetup newSetup, QString theme)
{
    setupSyntaxHighlighter(newSetup, theme);
    rehighlight();
}

void SyntaxHighlighter::setLangRules()
{
    if (setup == HighlightingSetup::none) {
        highlightingRules = {};
        return;
    }

    HighlightingRule rule;

    /* Functions highlighting rules */
    functionFormat.setForeground(functionsColor);
    rule.pattern = QRegularExpression("\\b[A-Za-z0-9_]+(?=\\()");
    rule.format = functionFormat;
    highlightingRules.append(rule);

    keywordFormat.setForeground(keywordColor);
    keywordFormat.setFontWeight(QFont::Bold);

    QStringList keywordPatterns;
    keywordPatterns << "\\bchar\\b" << "\\bint\\b" << "\\bfloat\\b" << "\\
bdouble\\b"
                    << "\\bstruct\\b" << "\\benum\\b" << "\\bvoid\\b" << "\\bshort\\b"
                    << "\\blong\\b" << "\\btrue\\b" << "\\bfalse\\b" << "\\bboolean\\b"
                    << "\\bnull\\b" << "\\bthis\\b" << "\\bfinal\\b"
                    << "\\band\\b" << "\\bor\\b" << "\\bxor\\b"
                    << "\\bconst\\b" << "\\bstatic\\b" << "\\bsigned\\b" << "\\bunsigned\\b"
                    << "\\bimport\\b" << "\\bnamespace\\b" << "\\breturn\\b" << "\\busing\\b"
                    << "\\bfor\\b" << "\\bwhile\\b" << "\\bif\\b" << "\\belse\\b"
                    << "\\bcase\\b" << "\\bswitch\\b" << "\\bdo\\b" << "\\bunion\\b"
                    << "\\bnew\\b" << "\\bclass\\b" << "\\bprivate\\b" << "\\bprotected\\b"
                    << "\\bpublic\\b" << "\\bvirtual\\b" << "\\bslots\\b" << "\\bvolatile\\b"
                    << "\\babstract\\b" << "\\bextends\\b" << "\\bimplements\\b" << "\\bsuper\\b"
                    << "\\btemplate\\b" << "\\btypedef\\b" << "\\btypename\\b"
                    << "\\btry\\b" << "\\bcatch\\b" << "\\bthrow\\b" << "\\bbreak\\b";
    for (const QString &pattern : keywordPatterns) {
        rule.pattern = QRegularExpression(pattern);
        rule.format = keywordFormat;
        highlightingRules.append(rule);
    }

    numberFormat.setForeground(numColor);
    rule.pattern = QRegularExpression("\\b[0-9\\.]+\\b");
    rule.format = numberFormat;
    highlightingRules.append(rule);

    quotationFormat.setForeground(valueColor);
    rule.pattern = QRegularExpression("\".*\"");
    rule.format = quotationFormat;
    highlightingRules.append(rule);

    quotationFormat.setForeground(valueColor);
    rule.pattern = QRegularExpression("’.*’");
    rule.format = quotationFormat;
    highlightingRules.append(rule);

    formatStringFormat.setForeground(formatStringColor);
    rule.pattern = QRegularExpression("%[sdifFuoxXeEgGaAcpn]+\\b");
    rule.format = formatStringFormat;
    highlightingRules.append(rule);

    headerFileFormat.setForeground(QColor("#ff6d6d"));
    rule.pattern = QRegularExpression("#include.?[<\"].*[>\"]");
    rule.format = headerFileFormat;
    highlightingRules.append(rule);

    singleLineCommentFormat.setFontItalic(true);
    singleLineCommentFormat.setForeground(commentColor);
    rule.pattern = QRegularExpression("//[^\n]*");
    rule.format = singleLineCommentFormat;
    highlightingRules.append(rule);

    multiLineCommentFormat.setFontItalic(true);
    multiLineCommentFormat.setForeground(commentColor);

    commentStartExpression = QRegularExpression("/\\*");
    commentEndExpression = QRegularExpression("\\*/");
}

void SyntaxHighlighter::setupKeywordPatterns()
{
    #ifdef EXTRA_VERBOSE
        std::cerr << "setup = " << static_cast<size_t>(setup) << " (" << HighlightingSetups[static_cast<size_t>(setup)] << ")\n";
    #endif


    switch (setup) {
    case (HighlightingSetup::cpp99):
        difKeywordPatterns  << "\\bchar\\b" << "\\bint\\b" << "\\bfloat\\b" << "\\bdouble\\b"
                            << "\\bstruct\\b" << "\\benum\\b" << "\\bvoid\\b" << "\\bshort\\b"
                            << "\\blong\\b" << "\\btrue\\b" << "\\bfalse\\b" << "\\bboolean\\b"
                            << "\\bthis\\b" << "\\bfriend\\b" <<"\\bconstexpr\\b"

                            << "\\bconst\\b" << "\\bstatic\\b" << "\\bsigned\\b" << "\\bunsigned\\b"
                            << "\\bnamespace\\b" << "\\breturn\\b" << "\\busing\\b"
                            << "\\bfor\\b" << "\\bwhile\\b" << "\\bif\\b" << "\\belse\\b"
                            << "\\bcase\\b" << "\\bswitch\\b" << "\\bdo\\b" << "\\bunion\\b"

                            << "\\bnew\\b" << "\\bclass\\b" << "\\bprivate\\b" << "\\bprotected\\b"
                            << "\\bpublic\\b" << "\\bvirtual\\b" << "\\bextern\\b" << "\\bvolatile\\b"
                            << "\\btemplate\\b" << "\\btypedef\\b" << "\\btypename\\b"
                            << "\\btry\\b" << "\\bcatch\\b" << "\\bthrow\\b" << "\\bbreak\\b"
                            << "\\bgoto\\b" << "\\bregister\\b" << "\\
binline\\b"

                         << "\\band\\b" << "\\bbitor\\b" << "\\bor\\b"
<< "\\bxor\\b"
                         << "\\bor_eq\\b" << "\\band_eq\\b" << "\\
bbitand\\b" << "\\bcompl\\b"
                         << "\\bxor_eq\\b" << "\\bnot\\b" << "\\bnot_eq
\\b"

                         << "\\basm\\b" << "\\bauto\\b"
                         << "\\bbool\\b" << "\\bcontinue\\b"
                         << "\\bdefault\\b" << "\\bdelete\\b" << "\\
bdynamic_cast\\b"
                         << "\\bexplicit\\b" << "\\bexport\\b" << "\\
bmutable\\b"
                         << "\\boperator\\b" << "\\breinterpret_cast\\b"
 << "\\btypeid\\b"
                         << "\\bstatic_cast\\b" << "\\bwchar_t\\b" << "
\\bfinal\\b" << "\\boverride\\b";
     break;

 case (HighlightingSetup::cpp14):
      difKeywordPatterns << "\\bchar\\b" << "\\bint\\b" << "\\bfloat\\b"
 << "\\bdouble\\b"
                         << "\\bstruct\\b" << "\\benum\\b" << "\\bvoid\\
b" << "\\bshort\\b"
                         << "\\blong\\b" << "\\btrue\\b" << "\\bfalse\\b
" << "\\bboolean\\b"
                         << "\\bnullptr\\b" << "\\bthis\\b" << "\\
bfriend\\b" <<"\\bconstexpr\\b"

                         << "\\bconst\\b" << "\\bstatic\\b" << "\\
bsigned\\b" << "\\bunsigned\\b"
                         << "\\bnamespace\\b" << "\\breturn\\b" << "\\
busing\\b"
                         << "\\bfor\\b" << "\\bwhile\\b" << "\\bif\\b"
<< "\\belse\\b"
                         << "\\bcase\\b" << "\\bswitch\\b" << "\\bdo\\b"
 << "\\bunion\\b"

                         << "\\bnew\\b" << "\\bclass\\b" << "\\bprivate
\\b" << "\\bprotected\\b"
                         << "\\bpublic\\b" << "\\bvirtual\\b" << "\\
bextern\\b" << "\\bvolatile\\b"
                         << "\\btemplate\\b" << "\\btypedef\\b" << "\\
btypename\\b"
                         << "\\btry\\b" << "\\bcatch\\b" << "\\bthrow\\b
" << "\\bbreak\\b"
                         << "\\bgoto\\b" << "\\binline\\b" << "\\
bthread_local\\b"

                         << "\\band\\b" << "\\bbitor\\b" << "\\bor\\b"
<< "\\bxor\\b"
```

```
                         << "\\bor_eq\\b" << "\\band_eq\\b" << "\\
bbitand\\b" << "\\bcompl\\b"
                         << "\\bxor_eq\\b" << "\\bnot\\b" << "\\bnot_eq
\\b"

                         << "\\balignas\\b" << "\\balignof\\b" << "\\
basm\\b" << "\\bauto\\b"
                         << "\\bbool\\b" << "\\bchar16_t\\b" << "\\
bchar32_t\\b" << "\\bcontinue\\b"
                         << "\\bdecltype\\b" << "\\bdefault\\b" << "\\
bdelete\\b" << "\\bdynamic_cast\\b"
                         << "\\bexplicit\\b" << "\\bexport\\b" << "\\
bmutable\\b" << "\\bnoexcept\\b"
                         << "\\boperator\\b" << "\\breinterpret_cast\\b"
 << "\\btypeid\\b" << "\\bstatic_assert\\b"
                         << "\\bstatic_cast\\b" << "\\bwchar_t\\b" << "
\\bfinal\\b" << "\\boverride\\b"
                         << "\\bint8_t\\b" << "\\bint16_t\\b" << "\\
bint32_t\\b" << "\\bint64_t\\b"
                         << "\\bint_fast8_t\\b" << "\\bint_fast16_t\\b"
<< "\\bint_fast32_t\\b" << "\\bint_fast64_t\\b"
                         << "\\bint_least8_t\\b" << "\\bint_least16_t\\b
" << "\\bint_least32_t\\b" << "\\bint_least64_t\\b"
                         << "\\buint8_t\\b" << "\\buint16_t\\b" << "\\
buint32_t\\b" << "\\buint64_t\\b"
                         << "\\buint_fast8_t\\b" << "\\buint_fast16_t\\b
" << "\\buint_fast32_t\\b" << "\\buint_fast64_t\\b"
                         << "\\buint_least8_t\\b" << "\\buint_least16_t
\\b" << "\\buint_least32_t\\b" << "\\buint_least64_t\\b"
                         << "\\bintmax_t\\b" << "\\bwintptr_t\\b" << "\\
buintmax_t\\b" << "\\buintptr_t\\b";
     break;

 case (HighlightingSetup::cpp20):
      difKeywordPatterns << "\\bchar\\b" << "\\bint\\b" << "\\bfloat\\b"
 << "\\bdouble\\b"
                         << "\\bstruct\\b" << "\\benum\\b" << "\\bvoid\\
b" << "\\bshort\\b"
                         << "\\blong\\b" << "\\btrue\\b" << "\\bfalse\\b
" << "\\bboolean\\b"
                         << "\\bnullptr\\b" << "\\bthis\\b" << "\\
bfriend\\b" <<"\\bconstexpr\\b"

                         << "\\bconst\\b" << "\\bstatic\\b" << "\\
bsigned\\b" << "\\bunsigned\\b"
                         << "\\bnamespace\\b" << "\\breturn\\b" << "\\
busing\\b"
                         << "\\bfor\\b" << "\\bwhile\\b" << "\\bif\\b"
<< "\\belse\\b"
                         << "\\bcase\\b" << "\\bswitch\\b" << "\\bdo\\b"
 << "\\bunion\\b"

                         << "\\bnew\\b" << "\\bclass\\b" << "\\bprivate
```

```
\\b" << "\\bprotected\\b"
                         << "\\bpublic\\b" << "\\bvirtual\\b" << "\\
bextern\\b" << "\\bvolatile\\b"
                         << "\\btemplate\\b" << "\\btypedef\\b" << "\\
btypename\\b"
                         << "\\btry\\b" << "\\bcatch\\b" << "\\bthrow\\b
" << "\\bbreak\\b"
                         << "\\bgoto\\b"<< "\\binline\\b"<< "\\
bthread_local\\b"

                         << "\\balignas\\b" << "\\balignof\\b" << "\\
basm\\b" << "\\bauto\\b"
                         << "\\bbool\\b" << "\\bchar16_t\\b" << "\\
bchar32_t\\b" << "\\bcontinue\\b"
                         << "\\bdecltype\\b" << "\\bdefault\\b" << "\\
bdelete\\b" << "\\bdynamic_cast\\b"
                         << "\\bexplicit\\b" << "\\bexport\\b" << "\\
bmutable\\b" << "\\bnoexcept\\b"
                         << "\\boperator\\b" << "\\breinterpret_cast\\b"
 << "\\btypeid\\b" << "\\bstatic_assert\\b"
                         << "\\bstatic_cast\\b" << "\\bwchar_t\\b" << "
\\bfinal\\b" << "\\boverride\\b"

                         << "\\band\\b" << "\\bbitor\\b" << "\\bor\\b"
<< "\\bxor\\b"
                         << "\\bor_eq\\b" << "\\band_eq\\b" << "\\
bbitand\\b" << "\\bcompl\\b"
                         << "\\bxor_eq\\b" << "\\bnot\\b" << "\\bnot_eq
\\b"

                         // C++20:
                         << "\\bchar8_t \\b" << "\\bconcept\\b" << "\\
bconsteval\\b" << "\\bconstinit\\b"
                         << "\\bco_await\\b" << "\\bco_return \\b" << "
\\bco_yield\\b" << "\\brequires\\b"
                         << "\\bstatic_cast\\b" << "\\bwchar_t\\b" << "
\\bfinal\\b" << "\\boverride\\b"
                         << "\\bstatic_cast\\b" << "\\bwchar_t\\b" << "
\\bfinal\\b" << "\\boverride\\b";
     break;

 case (HighlightingSetup::c11):
     difKeywordPatterns  << "\\bchar\\b" << "\\bint\\b" << "\\bfloat\\b"
 << "\\bdouble\\b"
                         << "\\bstruct\\b" << "\\benum\\b" << "\\bvoid\\
b" << "\\bshort\\b"
                         << "\\blong\\b"
                         << "\\bconst\\b" << "\\bstatic\\b" << "\\
bsigned\\b" << "\\bunsigned\\b"
                         << "\\breturn\\b"
                         << "\\bfor\\b" << "\\bwhile\\b" << "\\bif\\b"
<< "\\belse\\b"
                         << "\\bcase\\b" << "\\bswitch\\b" << "\\bdo\\b"
 << "\\bunion\\b"
                         << "\\bvolatile\\b" <<"\\bextern\\b" << "\\bgoto\\b" << "\\bregister\\b"
                         << "\\btypedef\\b" <<"\\bsizeof\\b" << "\\brestrict\\b" << "\\binline\\b"
                         << "\\band\\b" << "\\bbitor\\b" << "\\bor\\b" << "\\bxor\\b"
                         << "\\bor_eq\\b" << "\\band_eq\\b" << "\\bbitand\\b" << "\\bcompl\\b"
                         << "\\bxor_eq\\b" << "\\bnot\\b" << "\\bnot_eq\\b"
                            // from C99:
                         << "\\brestrict\\b" << "\\binline\\b"
                         << "\\b_Complex\\b" << "\\b_Bool\\b" << "\\b_Imaginary\\b"

                         << "\\bint8_t\\b" << "\\bint16_t\\b" << "\\bint32_t\\b" << "\\bint64_t\\b"
                         << "\\bint_fast8_t\\b" << "\\bint_fast16_t\\b" << "\\bint_fast32_t\\b" << "\\bint_fast64_t\\b"
                         << "\\bint_least8_t\\b" << "\\bint_least16_t\\b" << "\\bint_least32_t\\b" << "\\bint_least64_t\\b"
                         << "\\buint8_t\\b" << "\\buint16_t\\b" << "\\buint32_t\\b" << "\\buint64_t\\b"
                         << "\\buint_fast8_t\\b" << "\\buint_fast16_t\\b" << "\\buint_fast32_t\\b" << "\\buint_fast64_t\\b"
                         << "\\buint_least8_t\\b" << "\\buint_least16_t\\b" << "\\buint_least32_t\\b" << "\\buint_least64_t\\b"
                         << "\\bintmax_t\\b" << "\\bwintptr_t\\b" << "\\buintmax_t\\b" << "\\buintptr_t\\b";
     break;

 case (HighlightingSetup::c18):
      difKeywordPatterns << "\\bchar\\b" << "\\bint\\b" << "\\bfloat\\b" << "\\bdouble\\b"
                         << "\\bstruct\\b" << "\\benum\\b" << "\\bvoid\\b" << "\\bshort\\b"
                         << "\\blong\\b"
                         << "\\bconst\\b" << "\\bstatic\\b" << "\\bsigned\\b" << "\\bunsigned\\b"
                         << "\\breturn\\b"

                         << "\\band\\b" << "\\bbitor\\b" << "\\bor\\b" << "\\bxor\\b"
                         << "\\bor_eq\\b" << "\\band_eq\\b" << "\\bbitand\\b" << "\\bcompl\\b"
                         << "\\bxor_eq\\b" << "\\bnot\\b" << "\\bnot_eq\\b"

                         << "\\bfor\\b" << "\\bwhile\\b" << "\\bif\\b" << "\\belse\\b"
                         << "\\bcase\\b" << "\\bswitch\\b" << "\\bdo\\b" << "\\bunion\\b"
```

```
                            << "\\bvolatile\\b" <<"\\bextern\\b" << "\\bgoto\\b" << "\\binline\\b"
                            << "\\btypedef\\b" <<"\\bsizeof\\b" << "\\brestrict\\b" << "\\bregister\\b"
                            // from C18:
                            << "\\b_Alignas\\b" << "\\b_Alignof\\b" << "\\b_Atomic\\b" << "\\b_Bool\\b"
                            << "\\b_Complex\\b" << "\\b_Generic\\b" << "\\b_Imaginary\\b" << "\\b_Noreturn\\b"
                            << "\\b_Static_assert\\b" << "\\b_Thread_local\\b";
        break;

    case (HighlightingSetup::none):
        break;
    }
}

void SyntaxHighlighter::setColorValues(QString theme)
{
    if (theme == "monokai"){
        // monokai
        commonTextColorIsWhite = 1;
        keywordColor = QColor(102, 217, 239);
        keyword2Color = QColor(249, 38, 114);
        functionsColor = QColor(166, 226, 46);
        valueColor = QColor(230, 218, 117);
        numColor = QColor(174, 130, 255);
        operatorColor = QColor(249, 38, 114);
        formatStringColor = QColor(174, 130, 255);
        commentColor = QColor(178, 179, 191);
        varColor = QColor(102, 217, 239);
        tagColor = QColor(249, 38, 114);
        htmlAttributesColor = QColor(166, 226, 46);
        cssClassesIDsColor = QColor(166, 226, 46);
        cssAttributesColor = QColor(102, 217, 239);

    } else if (theme == "tomorrow"){
        // tomorrow
        commonTextColorIsWhite = 0;
        keywordColor = QColor(135, 88, 166);
        keyword2Color = QColor(135, 88, 166);
        functionsColor = QColor(66, 114, 173);
        valueColor = QColor(112, 138, 0);
        numColor = QColor(245, 135, 32);
        operatorColor = QColor(77, 77, 76);
        formatStringColor = QColor(199, 40, 40);
        commentColor = QColor(144, 143, 140);
        varColor = QColor(199, 40, 40);
        tagColor = QColor(199, 40, 40);
        htmlAttributesColor = QColor(245, 135, 32);
        cssClassesIDsColor = QColor(62, 153, 158);
        cssAttributesColor = QColor(77, 77, 76);

    } else if (theme == "tomorrowNight"){
        // tomorrow night
        commonTextColorIsWhite = 1;
        keywordColor = QColor(177, 149, 186);
        keyword2Color = QColor(177, 149, 186);
        functionsColor = QColor(128, 162, 189);
        valueColor = QColor(182, 189, 106);
        numColor = QColor(222, 146, 95);
        operatorColor = QColor(197, 199, 198);
        formatStringColor = QColor(222, 146, 95);
        commentColor = QColor(149, 150, 149);
        varColor = QColor(204, 102, 102);
        tagColor = QColor(204, 102, 102);
        htmlAttributesColor = QColor(222, 146, 95);
        cssClassesIDsColor = QColor(138, 189, 181);
        cssAttributesColor = QColor(197, 199, 198);

    } else if (theme == "solarized") {
        // Solarized
        commonTextColorIsWhite = 0;
        keywordColor = QColor(181, 137, 0);
        keyword2Color = QColor(133, 153, 0);
        functionsColor = QColor(88, 110, 117);
        valueColor = QColor(42, 161, 152);
        numColor = QColor(42, 161, 152);
        operatorColor = QColor(181, 137, 0);
        formatStringColor = QColor(220, 50, 47);
        commentColor = QColor(178, 179, 191);
        varColor = QColor(38, 139, 210);
        tagColor = QColor(38, 139, 210);
        htmlAttributesColor = QColor(181, 137, 0);
        cssClassesIDsColor = QColor(133, 153, 0);
        cssAttributesColor = QColor(77,171,171);
    } else {
        std::cerr << "ERROR: unknown theme provided to SyntaxHighlighter!\n
   ";
    }
}

void SyntaxHighlighter::highlightBlock(const QString &text)
{
    for (const HighlightingRule &rule : qAsConst(highlightingRules)) {
        QRegularExpressionMatchIterator matchIterator = rule.pattern.
   globalMatch(text);
        while (matchIterator.hasNext()) {
            QRegularExpressionMatch match = matchIterator.next();
            setFormat(match.capturedStart(), match.capturedLength(), rule.
   format);
        }
    }

    setCurrentBlockState(0);
```

```

  if (commentStartExpression != QRegularExpression("") && commentEndExpression != QRegularExpression("")) {
    int startIndex = 0;
    if (previousBlockState() != 1)
      startIndex = text.indexOf(commentStartExpression);

    while (startIndex >= 0) {
      QRegularExpressionMatch match = commentEndExpression.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, multiLineCommentFormat);
      startIndex = text.indexOf(commentStartExpression, startIndex + commentLength);
    }
  }
}
```

Приложение Д

Д.1 Файл TextEditor.hpp

```
#pragma once
#include <QPlainTextEdit>
#include <QPainter>
#include <QTextBlock>

class CodeEditor : public QPlainTextEdit
{
    Q_OBJECT

public:
    CodeEditor(QWidget *parent = nullptr);

    void lineNumberAreaPaintEvent(QPaintEvent *event);
    int  lineNumberAreaWidth();

    bool hideLineNumber;
    void hideLineNumberArea()
    {
        hideLineNumber = true;
    }

    void showLineNumberArea()
    {
       hideLineNumber = false;
    }

protected:
    void resizeEvent(QResizeEvent *event) override;

private slots:
    void updateLineNumberAreaWidth();
    void highlightCurrentLine();
    void updateLineNumberArea(const QRect &rect, int dy);

private:
    QWidget *lineNumberArea;

friend class MainWindow;
};

class LineNumberArea : public QWidget
{
public:
    LineNumberArea(CodeEditor *editor) : QWidget(editor), codeEditor(editor
  )
    {}

    QSize sizeHint() const override
    {
        return QSize(codeEditor->lineNumberAreaWidth(), 0);
    }

protected:
    void paintEvent(QPaintEvent *event) override
    {
        codeEditor->lineNumberAreaPaintEvent(event);
    }

private:
    CodeEditor *codeEditor;
};
```

Д.2 Файл TextEditor.cpp

```
#include "TextEditor.hpp"

CodeEditor::CodeEditor(QWidget *parent) : QPlainTextEdit(parent)
{
    hideLineNumber = false;
    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();
    highlightCurrentLine();
}

int CodeEditor::lineNumberAreaWidth()
{
    if (hideLineNumber)
        return 0;

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

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

    return space;
}

void CodeEditor::updateLineNumberAreaWidth()
{
    setViewportMargins(lineNumberAreaWidth(), 0, 0, 0);
```

```cpp
}

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();
}

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

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

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

    if (!isReadOnly()) {
        QTextEdit::ExtraSelection selection;

        QColor lineColor = QColor(Qt::yellow).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::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 = qRound(blockBoundingRect(block).height()) + top;
    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 = qRound(blockBoundingRect(block).height()) + top;
        ++blockNumber;
    }
}
```