# Приложение В

# Листинг программы

|  |
| --- |
| mainwindow.h |
| #ifndef MAINWINDOW\_H  #define MAINWINDOW\_H  #include <QMainWindow>  #include <QDebug>  #include <Models/document.h>  #include <QDesktopServices>  #include <QUrl>  #include<QTabWidget>  #include <QToolBar>  #include <QMimeData>  #include <QUrl>  #include <QLocale>  #include <QTranslator>  #include <QCloseEvent>  #include "Models/codeeditor.h"  #include "Models/lexerphp.h"  #include "Models/styletheme.h"  #include "Models/lexicalscanner.h"  QT\_BEGIN\_NAMESPACE  namespace Ui {  class MainWindow;  }  QT\_END\_NAMESPACE  class MainWindow : public QMainWindow  {  Q\_OBJECT  public:  MainWindow(QWidget \*parent = nullptr);  ~MainWindow();  private slots:  void createDocument();  void openDocument();  void saveDocument();  void saveAsDocument();  void exitDocument();  void ovverideEditing();  void copyEditing();  void cutEditing();  void insertEditing();  void removeEditing();  void replaceEditing();  void selectAllEditing();  void callReference();  void aboutReference();  void openSettings();  void closeTab(int index);  void addNewTab();  void closeCurrentTab();  void updateCursorPosition();  void updateTabName(int index);  CodeEditor\* getCurrentIdTextEdit();  CodeEditor\* getTextEdit(int id);  void changeLanguage();  void changeTheme();  void onTextChanged();  void command();  void customizeFont();  bool checkWork();  protected:  void wheelEvent(QWheelEvent \*event) override;  void dragEnterEvent(QDragEnterEvent \*event) override;  void dropEvent(QDropEvent \*event) override;  void closeEvent(QCloseEvent \*event) override;  private:  Ui::MainWindow \*ui;  int current\_id\_;  Document \*document\_;  LexerPhp \*lexer\_;  StyleTheme \*theme\_;  bool isModified\_;  bool language\_ = true;  };  #endif // MAINWINDOW\_H |

|  |
| --- |
| mainwindow.cpp |
| #include "mainwindow.h"  #include "ui\_mainwindow.h"  MainWindow::MainWindow(QWidget \*parent)  : QMainWindow(parent)  , ui(new Ui::MainWindow)  {  setAcceptDrops(true);  ui->setupUi(this);  document\_ = new Document("new\_document.txt");  theme\_ = new StyleTheme();  lexer\_ = new LexerPhp(this);  ui->tableWidget->horizontalHeader()->setSectionResizeMode(QHeaderView::Stretch);  QTabWidget \*tabWidget = ui->tabWidget;  ui->tabWidget->setTabsClosable(true);  ui->tabWidget->setMovable(true);  connect(tabWidget, &QTabWidget::tabCloseRequested, this, &MainWindow::closeTab);  current\_id\_ = 0;  ui->createDocumentButton->setToolTip("Создать документ");  ui->openDocumentButton->setToolTip("Открыть документ");  ui->insertButton->setToolTip("Вставить текст");  ui->copyButton->setToolTip("Скопировать текст");  ui->ovverideButton->setToolTip("Отменить последнее изменение");  ui->replaceButton->setToolTip("Повторить последнее изменение");  ui->saveDocumentButton->setToolTip("Сохранить документ");  ui->cutButton->setToolTip("Вырезать текст");  connect(ui->create, &QAction::triggered, this, &MainWindow::createDocument);  connect(ui->open, &QAction::triggered, this, &MainWindow::openDocument);  connect(ui->save, &QAction::triggered, this, &MainWindow::saveDocument);  connect(ui->saveAs, &QAction::triggered, this, &MainWindow::saveAsDocument);  connect(ui->exit, &QAction::triggered, this, &MainWindow::exitDocument);  connect(ui->ovveride,&QAction::triggered,this,&MainWindow::ovverideEditing);  connect(ui->copy,&QAction::triggered,this,&MainWindow::copyEditing);  connect(ui->cut,&QAction::triggered,this,&MainWindow::cutEditing);  connect(ui->insert,&QAction::triggered,this,&MainWindow::insertEditing);  connect(ui->remove,&QAction::triggered,this,&MainWindow::removeEditing);  connect(ui->replace,&QAction::triggered,this,&MainWindow::replaceEditing);  connect(ui->select\_all,&QAction::triggered,this,&MainWindow::selectAllEditing);  connect(ui->callReference,&QAction::triggered,this,&MainWindow::callReference);  connect(ui->aboutReference,&QAction::triggered,this,&MainWindow::aboutReference);  connect(ui->createDocumentButton,&QPushButton::clicked,this,&MainWindow::createDocument);  connect(ui->openDocumentButton,&QPushButton::clicked,this,&MainWindow::openDocument);  connect(ui->saveDocumentButton,&QPushButton::clicked,this,&MainWindow::saveDocument);  connect(ui->ovverideButton,&QPushButton::clicked,this,&MainWindow::ovverideEditing);  connect(ui->replaceButton,&QPushButton::clicked,this,&MainWindow::replaceEditing);  connect(ui->copyButton,&QPushButton::clicked,this,&MainWindow::copyEditing);  connect(ui->cutButton,&QPushButton::clicked,this,&MainWindow::cutEditing);  connect(ui->insertButton,&QPushButton::clicked,this,&MainWindow::insertEditing);  connect(ui->command, &QMenu::aboutToShow, this, &MainWindow::command);  connect(ui->changeLang,&QAction::triggered,this,&MainWindow::changeLanguage);  connect(ui->changeTheme,&QAction::triggered,this,&MainWindow::changeTheme);  connect(ui->customizeFont,&QAction::triggered,this,&MainWindow::customizeFont);  }  MainWindow::~MainWindow()  {  delete ui;  }  void MainWindow::createDocument(){  addNewTab();  document\_->create(getTextEdit(current\_id\_));  updateTabName(current\_id\_);  }  void MainWindow::openDocument(){  addNewTab();  document\_->open(getTextEdit(current\_id\_));  updateTabName(current\_id\_);  }  void MainWindow::saveDocument(){  if(!checkWork()) return;  document\_->save(getTextEdit(current\_id\_));  closeTab(current\_id\_);  }  void MainWindow::saveAsDocument(){  if(!checkWork()) return;  addNewTab();  document\_->saveAs(getTextEdit(current\_id\_));  updateTabName(current\_id\_);  }  void MainWindow::exitDocument(){  document\_->exit(getTextEdit(current\_id\_));  QApplication::quit();  }  void MainWindow::ovverideEditing(){  if(!checkWork()) return;  getCurrentIdTextEdit()->undo();  }  void MainWindow::copyEditing(){  if(!checkWork()) return;  getCurrentIdTextEdit()->copy();  }  void MainWindow::cutEditing(){  if(!checkWork()) return;  getCurrentIdTextEdit()->cut();  }  void MainWindow::insertEditing(){  if(!checkWork()) return;  getCurrentIdTextEdit()->paste();  }  void MainWindow::removeEditing(){  if(!checkWork()) return;  getCurrentIdTextEdit()->textCursor().removeSelectedText();  }  void MainWindow::replaceEditing(){  if(!checkWork()) return;  getCurrentIdTextEdit()->redo();  }  void MainWindow::selectAllEditing(){  if(!checkWork()) return;  getCurrentIdTextEdit()->selectAll();  }  void MainWindow::callReference(){  QString path = QCoreApplication::applicationDirPath() + "/documentation/documentation.html";  QDesktopServices::openUrl(QUrl::fromLocalFile(path));  }  void MainWindow::aboutReference(){  QMessageBox::information(this,  "О программе",  "Приложение создана студентом НГТУ Шаталов Максимом.\n"  "Приложение создана для того, что бы запускать функции инициализации константы define() языка PHP.\n"  "О установке и управлении приложения можно почитать в документации.");  }  void MainWindow::openSettings(){  }  void MainWindow::wheelEvent(QWheelEvent \*event){  CodeEditor\* editor = getCurrentIdTextEdit();  if(event->modifiers() & Qt::ControlModifier){  int delta = event->angleDelta().y();  QFont fontTextDocument = editor->font();  int newSize = fontTextDocument.pointSize() + (delta > 0 ? 1 : -1);  if (newSize >= 8 && newSize <= 32) {  fontTextDocument.setPointSize(newSize);  editor->setFont(fontTextDocument);  }  event->accept();  }  }  void MainWindow::closeTab(int index){  QWidget \*widget = ui->tabWidget->widget(index);  ui->tabWidget->removeTab(index);  delete widget;  }  void MainWindow::addNewTab(){  CodeEditor \*textEdit = new CodeEditor(this);  lexer\_->setDocument(textEdit->document());  connect(textEdit, &QPlainTextEdit::textChanged, this, &MainWindow::onTextChanged);  current\_id\_ = ui->tabWidget->addTab(textEdit,"Новый документ");  ui->tabWidget->setCurrentIndex(current\_id\_);  }  void MainWindow::closeCurrentTab(){  closeTab(ui->tabWidget->currentIndex());  }  void MainWindow::dragEnterEvent(QDragEnterEvent \*event){  if (event->mimeData()->hasUrls()) {  event->acceptProposedAction();  }  }  void MainWindow::dropEvent(QDropEvent \*event){  QList<QUrl> urls = event->mimeData()->urls();  if (urls.isEmpty()) return;  QString filePath = urls.first().toLocalFile();  CodeEditor \*editor = qobject\_cast<CodeEditor\*>(ui->tabWidget->widget(current\_id\_));  document\_->open(filePath,editor);  }  CodeEditor\* MainWindow::getCurrentIdTextEdit(){  return qobject\_cast<CodeEditor\*>(ui->tabWidget->widget(ui->tabWidget->currentIndex()));  }  CodeEditor\* MainWindow::getTextEdit(int id){  return qobject\_cast<CodeEditor\*>(ui->tabWidget->widget(id));  }  void MainWindow::updateCursorPosition(){  }  void MainWindow::updateTabName(int index){  QString fileName = document\_->getFileName();  if (!fileName.isEmpty()) {  ui->tabWidget->setTabText(index, fileName);  }  }  void MainWindow::closeEvent(QCloseEvent \*event){  if (isModified\_) {  QMessageBox::StandardButton resBtn = QMessageBox::question(this, "Программа",  "Сохранить изменения?", QMessageBox::Save | QMessageBox::Discard | QMessageBox::Cancel,  QMessageBox::Save);  if (resBtn == QMessageBox::Save) {  saveDocument();  event->accept();  } else if (resBtn == QMessageBox::Discard) {  event->accept();  } else {  event->ignore();  }  } else {  event->accept();  }  }  void MainWindow::onTextChanged(){  isModified\_ = true;  }  void MainWindow::changeLanguage(){  language\_ = language\_ ? false : true;  if(language\_){  ui->menuFile->setTitle("Файл");  ui->create->setText("Создать");  ui->open->setText("Открыть");  ui->save->setText("Сохранить");  ui->saveAs->setText("Сохранить как");  ui->exit->setText("Выход");  ui->menuFile->setTitle("Правка");  ui->ovveride->setText("Отменить");  ui->replace->setText("Повторить");  ui->cut->setText("Вырезать");  ui->copy->setText("Копировать");  ui->insert->setText("Вставить");  ui->remove->setText("Удалить");  ui->select\_all->setText("Выделить все");  ui->menuText->setTitle("Текст");  ui->analysisMethod->setText("Метод анализа");  ui->classificationGrammar->setText("Классификация грамматики");  ui->diagnosticsAnalysisError->setText("Диагностика и нейтрализация ошибок");  ui->grammar->setText("Грамматика");  ui->references->setText("Список литературы");  ui->sourceCodeProgramm->setText("Исходный код программы");  ui->statmentProblem->setText("Постановка задачи");  ui->testCase->setText("Тестовый пример");  ui->settings->setTitle("Настройки");  ui->customizeFont->setText("Настроить шрифт");  ui->changeLang->setText("Настроить язык");  ui->changeTheme->setText("Сменить тему");  ui->reference->setTitle("Справка");  ui->callReference->setText("Вызов справки");  ui->aboutReference->setText("О программе");  ui->command->setTitle("Пуск");  QStringList headers = {"Путь к файлу", "Линия", "Сообщение"};  ui->tableWidget->setHorizontalHeaderLabels(headers);  }  else{  ui->menuFile->setTitle("File");  ui->create->setText("Create");  ui->open->setText("Open");  ui->save->setText("Save");  ui->saveAs->setText("Save as");  ui->exit->setText("Exit");  ui->menuEditor->setTitle("Editing");  ui->ovveride->setText("Ovveride");  ui->replace->setText("Replace");  ui->cut->setText("Cut");  ui->copy->setText("Copy");  ui->insert->setText("Insert");  ui->remove->setText("Remove");  ui->select\_all->setText("Select all");  ui->menuText->setTitle("Text");  ui->analysisMethod->setText("Analysis method");  ui->classificationGrammar->setText("Classification grammar");  ui->diagnosticsAnalysisError->setText("Diagnostics and analysis error");  ui->grammar->setText("Grammar");  ui->references->setText("References");  ui->sourceCodeProgramm->setText("Source code programm");  ui->statmentProblem->setText("Statment problem");  ui->testCase->setText("Test case");  ui->settings->setTitle("Settings");  ui->customizeFont->setText("Customize font");  ui->changeLang->setText("Change language");  ui->changeTheme->setText("Change theme");  ui->reference->setTitle("Reference");  ui->callReference->setText("Call reference");  ui->aboutReference->setText("About programm");  ui->command->setTitle("Command");  QStringList headers = {"File path", "Line", "Message"};  ui->tableWidget->setHorizontalHeaderLabels(headers);  }  }  void MainWindow::customizeFont(){  QMessageBox::information(this,  "Изменения шрифта",  "ctrl+MouseWheel");  }  bool MainWindow::checkWork(){  if(!current\_id\_) return true;  else return false;  }  void MainWindow::changeTheme(){  if(!theme\_->theme\_){  setStyleSheet(theme\_->ligthTheme\_);  theme\_->theme\_ = true;  }  else{  setStyleSheet(theme\_->darkTheme\_);  theme\_->theme\_ = false;  }  }  void MainWindow::command(){  LexicalScanner lexer(getCurrentIdTextEdit()->toPlainText());  lexer.analyzeToTable(ui->tableWidget);  } |

|  |
| --- |
| lexicalscanner.h |
| #ifndef LEXICALSCANNER\_H  #define LEXICALSCANNER\_H  #include <QString>  #include <QTableWidget>  #include <QChar>  // Типы токенов  enum TokenType {  TOKEN\_UNKNOWN, // Неизвестный токен  TOKEN\_DEFINE, // Ключевое слово define  TOKEN\_IDENTIFIER, // Идентификатор (имя константы)  TOKEN\_STRING, // Строка  TOKEN\_INT, // Целое число  TOKEN\_FLOAT, // Число с плавающей точкой  TOKEN\_LPAREN, // Открывающая скобка '('  TOKEN\_RPAREN, // Закрывающая скобка ')'  TOKEN\_COMMA, // Запятая ','  TOKEN\_SEMICOLON, // Точка с запятой ';'  TOKEN\_END // Конец входных данных  };  struct Token {  TokenType type;  QString value;  int lineNumber;  int startPos;  int endPos;  };  class LexicalScanner {  public:  LexicalScanner(const QString &input);  Token getNextToken();  void analyzeToTable(QTableWidget \*table);  bool hasSyntaxError() const;  private:  void advance();  Token createToken(TokenType type, const QString &value, int start, int end);  bool peekKeyword(const QString &keyword);  Token readString(QChar quoteChar);  bool isRussianLetter(QChar ch);  QString input;  int position;  int lineNumber;  bool syntaxError;  bool foundDefine;  bool foundName;  bool foundComma;  bool foundValue;  bool foundRParen;  bool foundSemicolon;  };  #endif // LEXICALSCANNER\_H |

|  |
| --- |
| lexicalscanner.cpp |
| #include "lexicalscanner.h"  #include <stdexcept>  LexicalScanner::LexicalScanner(const QString &input)  : input(input), position(0), lineNumber(1), syntaxError(false), foundDefine(false), foundName(false), foundComma(false), foundValue(false), foundRParen(false), foundSemicolon(false) {}  void LexicalScanner::advance() {  position++;  }  Token LexicalScanner::createToken(TokenType type, const QString &value, int start, int end) {  return {type, value, lineNumber, start, end};  }  bool LexicalScanner::peekKeyword(const QString& keyword) {  int len = keyword.length();  if (input.mid(position, len) == keyword) {  if (position + len < input.length()) {  QChar nextChar = input[position + len];  if (nextChar.isLetterOrNumber() || nextChar == '\_') {  syntaxError = true;  return false;  }  }  return true;  }  return false;  }  Token LexicalScanner::readString(QChar quoteChar) {  int start = position;  QString str;  advance();  while (position < input.length() && input[position] != quoteChar) {  str += input[position];  advance();  }  if (position >= input.length() || input[position] != quoteChar) {  syntaxError = true;  return createToken(TOKEN\_UNKNOWN, "Ошибка: незакрытая строка", start, position);  }  advance();  return createToken(TOKEN\_STRING, str, start, position);  }  Token LexicalScanner::getNextToken() {  while (position < input.length()) {  QChar currentChar = input[position];  if (currentChar.isSpace()) {  advance();  continue;  }  if (peekKeyword("define")) {  int start = position;  position += 6;  foundDefine = true;  return createToken(TOKEN\_DEFINE, "define", start, position);  }  if (currentChar.isLetter() || currentChar == '\_') {  QString identifier;  int start = position;  while (position < input.length() && (input[position].isLetterOrNumber() || input[position] == '\_')) {  identifier += input[position];  advance();  }  // Устанавливаем foundName только если найдено define  if (foundDefine) {  foundName = true;  }  return createToken(TOKEN\_IDENTIFIER, identifier, start, position);  }  if (currentChar.isDigit()) {  QString number;  int start = position;  bool isFloat = false;  while (position < input.length() && (input[position].isDigit() || input[position] == '.')) {  if (input[position] == '.') {  isFloat = true;  }  number += input[position];  advance();  }  foundValue = true;  return createToken(isFloat ? TOKEN\_FLOAT : TOKEN\_INT, number, start, position);  }  if (currentChar == '(' || currentChar == ')' || currentChar == ',' || currentChar == ';') {  TokenType type;  switch (currentChar.unicode()) {  case '(': type = TOKEN\_LPAREN; break;  case ')': foundRParen = true; type = TOKEN\_RPAREN; break;  case ',': foundComma = true; type = TOKEN\_COMMA; break;  case ';': foundSemicolon = true; type = TOKEN\_SEMICOLON; break;  default: type = TOKEN\_UNKNOWN; break;  }  advance();  return createToken(type, QString(currentChar), position - 1, position);  }  if (currentChar == '"' || currentChar == '\'') {  Token token = readString(currentChar);  if (foundDefine && token.type == TOKEN\_STRING) {  foundName = true; // Устанавливаем флаг, если это строка и найдено define  }  return token;  }  advance();  return createToken(TOKEN\_UNKNOWN, QString("Ошибка: неизвестный символ '%1'").arg(currentChar), position - 1, position);  }  return {TOKEN\_END, "", lineNumber, position, position};  }  void LexicalScanner::analyzeToTable(QTableWidget\* table) {  table->setRowCount(0);  table->setColumnCount(1);  table->setHorizontalHeaderLabels({"Ошибки"});  Token token;  int errorCount = 0;  while ((token = getNextToken()).type != TOKEN\_END) {  if (token.type == TOKEN\_UNKNOWN && token.value.contains("Ошибка")) {  int row = table->rowCount();  table->insertRow(row);  table->setItem(row, 0, new QTableWidgetItem(token.value));  errorCount++;  }  }  if (!foundDefine) {  table->insertRow(table->rowCount());  table->setItem(table->rowCount() - 1, 0, new QTableWidgetItem("Ошибка: отсутствует ключевое слово define"));  errorCount++;  }  if (foundDefine && !foundName) {  table->insertRow(table->rowCount());  table->setItem(table->rowCount() - 1, 0, new QTableWidgetItem("Ошибка: отсутствует имя константы"));  errorCount++;  }  if (!foundComma) {  table->insertRow(table->rowCount());  table->setItem(table->rowCount() - 1, 0, new QTableWidgetItem("Ошибка: отсутствует запятая"));  errorCount++;  }  if (!foundValue) {  table->insertRow(table->rowCount());  table->setItem(table->rowCount() - 1, 0, new QTableWidgetItem("Ошибка: отсутствует значение константы"));  errorCount++;  }  if (!foundRParen) {  table->insertRow(table->rowCount());  table->setItem(table->rowCount() - 1, 0, new QTableWidgetItem("Ошибка: отсутствует закрывающая скобка"));  errorCount++;  }  if (!foundSemicolon) {  table->insertRow(table->rowCount());  table->setItem(table->rowCount() - 1, 0, new QTableWidgetItem("Ошибка: отсутствует завершающий оператор ;"));  errorCount++;  }  table->insertRow(table->rowCount());  table->setItem(table->rowCount() - 1, 0, new QTableWidgetItem(QString("Всего ошибок: %1").arg(errorCount)));  }  bool LexicalScanner::hasSyntaxError() const {  return syntaxError;  }  bool LexicalScanner::isRussianLetter(QChar ch) {  return (ch >= QChar(u'а')) && (ch <= QChar(u'я')) ||  (ch >= QChar(u'А')) && (ch <= QChar(u'Я'));  } |

|  |
| --- |
| lexerphp.h |
| #ifndef LEXERPHP\_H  #define LEXERPHP\_H  #include <QSyntaxHighlighter>  #include <QTextCharFormat>  class LexerPhp : public QSyntaxHighlighter  {  public:  explicit LexerPhp(QObject \*parent = nullptr);  protected:  void highlightBlock(const QString &text) override;  private:  QTextCharFormat keywordFormat;  QTextCharFormat commentFormat;  QTextCharFormat stringFormat;  void initFormats();  };  #endif // LEXERPHP\_H |

|  |
| --- |
| lexerphp.cpp |
| #include "lexerphp.h"  #include <QRegularExpression>  LexerPhp::LexerPhp(QObject \*parent)  : QSyntaxHighlighter(parent)  {  initFormats();  }  void LexerPhp::highlightBlock(const QString &text)  {  QRegularExpression keywordRegex("\\b(?:echo|if|else|while|for|function|return)\\b");  QRegularExpressionMatchIterator keywordIterator = keywordRegex.globalMatch(text);  while (keywordIterator.hasNext()) {  QRegularExpressionMatch match = keywordIterator.next();  setFormat(match.capturedStart(), match.capturedLength(), keywordFormat);  }  QRegularExpression commentRegex("//[^\n]\*");  QRegularExpressionMatchIterator commentIterator = commentRegex.globalMatch(text);  while (commentIterator.hasNext()) {  QRegularExpressionMatch match = commentIterator.next();  setFormat(match.capturedStart(), match.capturedLength(), commentFormat);  }  QRegularExpression stringRegex("\".\*?\"");  QRegularExpressionMatchIterator stringIterator = stringRegex.globalMatch(text);  while (stringIterator.hasNext()) {  QRegularExpressionMatch match = stringIterator.next();  setFormat(match.capturedStart(), match.capturedLength(), stringFormat);  }  }  void LexerPhp::initFormats()  {  keywordFormat.setForeground(Qt::blue);  keywordFormat.setFontWeight(QFont::Bold);  commentFormat.setForeground(Qt::green);  commentFormat.setFontItalic(true);  stringFormat.setForeground(Qt::darkRed);  } |

|  |
| --- |
| document.h |
| #ifndef DOCUMENT\_H  #define DOCUMENT\_H  #include <QFile>  #include <QPlainTextEdit>  #include <QFileDialog>  #include <QWidget>  #include <QTextStream>  #include <QMessageBox>  #include "Models/codeeditor.h"  #include <QFileInfo>  class Document  {  public:  Document(QString name\_file = "new\_document.txt");  void create(QPlainTextEdit\* editor);  void open(QPlainTextEdit\* editor);  void open(QString filePath,QPlainTextEdit\* editor);  void save(QPlainTextEdit\* editor);  void saveAs(QPlainTextEdit \*editor);  void exit(QPlainTextEdit\* editor);  QString getFileName();  private:  QString name\_file\_;  QFileInfo \*fileInfo;  QFile \*file\_;  };  #endif // DOCUMENT\_H |

|  |
| --- |
| document.cpp |
| #include "document.h"  Document::Document(QString name\_file){  file\_ = new QFile(name\_file);  }  void Document::create(QPlainTextEdit\* editor){  name\_file\_ = QFileDialog::getSaveFileName(nullptr,  "Сохранить файл",  QDir::homePath(),  "Текстовые файлы (\*.txt);;Все файлы (\*.\*)");  fileInfo = new QFileInfo(name\_file\_);  if (name\_file\_.isEmpty()) {  return;  }  this->file\_ = new QFile(name\_file\_);  if (!this->file\_->open(QIODevice::WriteOnly | QIODevice::Text)) {  QMessageBox::warning(nullptr,  "Ошибка",  "Не удалось создать файл");  return;  }  QTextStream out(this->file\_);  out << editor->toPlainText();  this->file\_->close();  }  void Document::open(QPlainTextEdit\* editor){  name\_file\_ = QFileDialog::getOpenFileName(new QWidget,  "Открыть файл",  "",  "Текстовые файлы (\*.txt);;Все файлы (\*)");  fileInfo = new QFileInfo(name\_file\_);  file\_ = new QFile(name\_file\_);  file\_->open(QIODevice::ReadOnly | QIODevice::Text);  QTextStream in(file\_);  QString file\_content = in.readAll();  editor->setPlainText(file\_content);  file\_->close();  }  void Document::open(QString filePath,QPlainTextEdit\* editor){\  name\_file\_ = filePath;  fileInfo = new QFileInfo(name\_file\_);  file\_ = new QFile(filePath);  file\_->open(QIODevice::ReadOnly | QIODevice::Text);  QTextStream in(file\_);  QString file\_content = in.readAll();  editor->setPlainText(file\_content);  file\_->close();  }  void Document::save(QPlainTextEdit\* editor){  file\_->open(QIODevice::WriteOnly | QIODevice::Text);  QTextStream out(file\_);  out << editor->toPlainText();  file\_->close();  }  void Document::saveAs(QPlainTextEdit\* editor){  name\_file\_ = QFileDialog::getSaveFileName(new QWidget,  "Сохранить файл",  "",  "Текстовые файлы (\*.txt);;Все файлы (\*)");  fileInfo = new QFileInfo(name\_file\_);  file\_ = new QFile(name\_file\_);  file\_->open(QIODevice::WriteOnly | QIODevice::Text);  QTextStream out(file\_);  out << editor->toPlainText();  file\_->close();  }  void Document::exit(QPlainTextEdit\* editor){  if(file\_->isOpen()){  file\_->close();  }  editor->clear();  }  QString Document::getFileName(){  return fileInfo->baseName();  } |

|  |
| --- |
| codeeditor.h |
| #ifndef CODEEDITOR\_H  #define CODEEDITOR\_H  #include <QPlainTextEdit>  #include <QWidget>  class LineNumberArea;  class CodeEditor : public QPlainTextEdit {  Q\_OBJECT  public:  explicit CodeEditor(QWidget \*parent = nullptr);  int lineNumberAreaWidth();  void lineNumberAreaPaintEvent(QPaintEvent \*event);  protected:  void resizeEvent(QResizeEvent \*event) override;  private slots:  void updateLineNumberAreaWidth();  void updateLineNumberArea(const QRect &rect, int dy);  private:  LineNumberArea \*lineNumberArea;  };  #endif // CODEEDITOR\_H |

|  |
| --- |
| codeeditor.h |
| #include "codeeditor.h"  #include "linenumberarea.h"  #include <QPainter>  #include <QTextBlock>  #include <Qsci/qsciscintilla.h>  CodeEditor::CodeEditor(QWidget \*parent) : QPlainTextEdit(parent) {  lineNumberArea = new LineNumberArea(this);  connect(this, &QPlainTextEdit::blockCountChanged, this, &CodeEditor::updateLineNumberAreaWidth);  connect(this, &QPlainTextEdit::updateRequest, this, &CodeEditor::updateLineNumberArea);  updateLineNumberAreaWidth();  }  int CodeEditor::lineNumberAreaWidth() {  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::resizeEvent(QResizeEvent \*event) {  QPlainTextEdit::resizeEvent(event);  QRect cr = contentsRect();  lineNumberArea->setGeometry(QRect(cr.left(), cr.top(), lineNumberAreaWidth(), cr.height()));  }  void CodeEditor::updateLineNumberAreaWidth() {  setViewportMargins(lineNumberAreaWidth(), 0, 0, 0);  }  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::lineNumberAreaPaintEvent(QPaintEvent \*event) {  QPainter painter(lineNumberArea);  painter.fillRect(event->rect(), Qt::lightGray);  QTextBlock block = firstVisibleBlock();  int blockNumber = block.blockNumber();  int top = (int) blockBoundingGeometry(block).translated(contentOffset()).top();  int bottom = top + (int) blockBoundingRect(block).height();  while (block.isValid() && top <= event->rect().bottom()) {  if (block.isVisible() && bottom >= event->rect().top()) {  QString number = QString::number(blockNumber + 1);  painter.setPen(Qt::black);  painter.drawText(0, top, lineNumberArea->width(), fontMetrics().height(),  Qt::AlignRight, number);  }  block = block.next();  top = bottom;  bottom = top + (int) blockBoundingRect(block).height();  ++blockNumber;  }  } |

|  |
| --- |
| linenumberarea.h |
| #include "linenumberarea.h"  #include "codeeditor.h"  #include <QPainter>  LineNumberArea::LineNumberArea(CodeEditor \*editor)  : QWidget(editor), textEditor(editor) {}  QSize LineNumberArea::sizeHint() const {  return QSize(textEditor->lineNumberAreaWidth(), 0);  }  void LineNumberArea::paintEvent(QPaintEvent \*event) {  textEditor->lineNumberAreaPaintEvent(event);  } |

|  |
| --- |
| linenumberarea.cpp |
| #include "linenumberarea.h"  #include "codeeditor.h"  #include <QPainter>  LineNumberArea::LineNumberArea(CodeEditor \*editor)  : QWidget(editor), textEditor(editor) {}  QSize LineNumberArea::sizeHint() const {  return QSize(textEditor->lineNumberAreaWidth(), 0);  }  void LineNumberArea::paintEvent(QPaintEvent \*event) {  textEditor->lineNumberAreaPaintEvent(event);  } |

|  |
| --- |
| styletheme.h |
| #ifndef STYLETHEME\_H  #define STYLETHEME\_H  #include <QString>  class StyleTheme  {  public:  StyleTheme();  bool theme\_;  QString darkTheme\_;  QString ligthTheme\_;  };  #endif // STYLETHEME\_H |

|  |
| --- |
| styletheme.cpp |
| #include "styletheme.h"  StyleTheme::StyleTheme() {  darkTheme\_ = R"(QPushButton {  background-color: #333;  color: white;  border-radius: 8px;  padding: 5px;  font-size: 14px;  }  QPushButton:hover {  background-color: #555;  }  QPushButton:pressed {  background-color: #777;  }  QTabWidget {  background-color: #2E2E2E;  color: #D8D8D8;  border: 2px solid #555555;  border-radius: 8px;  }  QTabWidget::pane {  border: none;  background-color: #2E2E2E;  }  QTabBar::tab {  background-color: #3C3C3C;  color: #D8D8D8;  padding: 10px;  border-radius: 5px;  min-width: 100px;  }  QTabBar::tab:selected {  background-color: #5C6BC0;  color: white;  border-radius: 5px;  }  QTabBar::tab:hover {  background-color: #4C4C4C;  color: white;  }  QTabBar::tab:pressed {  background-color: #6A7F93;  color: white;  }  QTableWidget {  background-color: #333333;  color: #D8D8D8;  border: 1px solid #555555;  }  QHeaderView::section {  background-color: #444444;  color: #D8D8D8;  padding: 5px;  border: 1px solid #555555;  }  QTableWidget::item {  background-color: #333333;  color: #D8D8D8;  border: 1px solid #555555;  }  QTableWidget::item:selected {  background-color: #5C6BC0;  color: white;  }  QTableWidget::horizontalHeader {  background-color: #444444;  }  QTableWidget::verticalHeader {  background-color: #444444;  }  QHeaderView::section {  background-color: #444444;  color: #D8D8D8;  padding: 8px;  border: 1px solid #555555;  border-radius: 5px;  }  QTableCornerButton::section {  background-color: #444444;  border: none;  }  QMainWindow {  background-color: #2E2E2E;  color: #D8D8D8;  border: 1px solid #555555;  border-radius: 8px;  }  QMenuBar {  background-color: #333333;  color: #D8D8D8;  border-bottom: 1px solid #555555;  }  QMenuBar::item {  background-color: #333333;  padding: 10px;  border-radius: 5px;  }  QMenuBar::item:selected {  background-color: #5C6BC0;  color: white;  }  QLineEdit {  background-color: #3C3C3C;  color: #D8D8D8;  border: 1px solid #555555;  border-radius: 5px;  padding: 5px;  }  QMessageBox {  background-color: #333333;  color: #D8D8D8;  border-radius: 10px;  font-family: Arial, sans-serif;  font-size: 14px;  }  QMessageBox QPushButton {  background-color: #444444;  color: #D8D8D8;  border: 1px solid #555555;  border-radius: 5px; /  padding: 5px 10px;  }  QMessageBox QPushButton:hover {  background-color: #5C6BC0;  color: white;  }  QMessageBox QPushButton:pressed {  background-color: #3F51B5;  color: white;  }  QMessageBox QLabel {  color: #D8D8D8;  }  )";  ligthTheme\_ = R"(QPushButton {  background-color: #F0F0F0;  color: #333333;  border-radius: 8px;  padding: 5px;  font-size: 14px;  }  QPushButton:hover {  background-color: #DDDDDD;  }  QPushButton:pressed {  background-color: #BBBBBB;  }  QTabWidget {  background-color: #FFFFFF;  color: #333333;  border: 2px solid #CCCCCC;  border-radius: 8px;  }  QTabWidget::pane {  border: none;  background-color: #FFFFFF;  }  QTabBar::tab {  background-color: #F5F5F5;  color: #333333;  padding: 10px;  border-radius: 5px;  min-width: 100px;  }  QTabBar::tab:selected {  background-color: #4CAF50;  color: white;  border-radius: 5px;  }  QTabBar::tab:hover {  background-color: #E0E0E0;  color: #333333;  }  QTabBar::tab:pressed {  background-color: #388E3C;  color: white;  }  QTableWidget {  background-color: #FFFFFF;  color: #333333;  border: 1px solid #CCCCCC;  }  QHeaderView::section {  background-color: #F1F1F1;  color: #333333;  padding: 5px;  border: 1px solid #CCCCCC;  }  QTableWidget::item {  background-color: #FFFFFF;  color: #333333;  border: 1px solid #CCCCCC;  }  QTableWidget::item:selected {  background-color: #4CAF50;  color: white;  }  QTableWidget::horizontalHeader {  background-color: #F1F1F1;  }  QTableWidget::verticalHeader {  background-color: #F1F1F1;  }  QHeaderView::section {  background-color: #F1F1F1;  color: #333333;  padding: 8px;  border: 1px solid #CCCCCC;  border-radius: 5px;  }  QTableCornerButton::section {  background-color: #F1F1F1;  border: none;  }  QMainWindow {  background-color: #FFFFFF;  color: #333333;  border: 1px solid #CCCCCC;  border-radius: 8px;  }  QMenuBar {  background-color: #F0F0F0;  color: #333333;  border-bottom: 1px solid #CCCCCC;  }  QMenuBar::item {  background-color: #F0F0F0;  padding: 10px;  border-radius: 5px;  }  QMenuBar::item:selected {  background-color: #4CAF50;  color: white;  }  QLineEdit {  background-color: #F9F9F9;  color: #333333;  border: 1px solid #CCCCCC;  border-radius: 5px;  padding: 5px;  }  QMessageBox {  background-color: #FFFFFF;  color: #333333;  border-radius: 10px;  font-family: Arial, sans-serif;  font-size: 14px;  }  QMessageBox QPushButton {  background-color: #F0F0F0;  color: #333333;  border: 1px solid #CCCCCC;  border-radius: 5px;  padding: 5px 10px;  }  QMessageBox QPushButton:hover {  background-color: #DDDDDD;  color: white;  }  QMessageBox QPushButton:pressed {  background-color: #BBBBBB;  color: white;  }  QMessageBox QLabel {  color: #333333;  }  )";  theme\_ = false;  } |