Правительство Российской Федерации

Федеральное государственное автономное образовательное учреждение

высшего образования

«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ “ВЫСШАЯ ШКОЛА ЭКОНОМИКИ”

Кафедра «Компьютерная безопасность»

ОТЧЕТ

К ЛАБОРАТОРНОЙ РАБОТЕ №12

По дисциплине

«Языки программирования»

Работу выполнил

Студент группы СКБ 193 _____ **С.О.Ташкинов**

дата,подпись

Работу проверил _____ **С.А. Булгаков**

Москва 2020

Оглавление

Постановка задачи

Разработать графическое приложение с использованием библиотеки Qt– текстовый редактор:
1)с подсветкой текущей строки;
2)с нумерацией строк;
3)с подсветкой синтаксиса (переключение): Си89, Си++98/03
Заголовок окна должен содержать имя редактируемого файла (ограниченное 32 символами + троеточие) и признак того что файл был изменен (звездочка в начале имени + курсив) с момента последнего сохранения.
Окно содержит главное меню (и наследуется от QMainWindow) состоящее из пунктов:
1)Файл [кнопки]
a)Новый
b)Открыть
c)Сохранить
d)Сохранить как
e)Выход
2)Правка[кнопки]
a)Отменить
b)Повторить
c)Копировать
d)Вырезать
e)Вставить
f)Найти
g)Найти и заменить
h)Выделить все
3)Формат
a)Перенос по словам [галочка]
b)Выбор шрифта -открывается модальный диалог выбора шрифта
4)Вид [галочки, где не указано иное]
a)Выбор цвета фона [кнопка] -открывает модальное окно выбора цвета
b)Выбор цвета текущей строки [кнопка] -открывает модальное окно выбора цвета
c)Вкл/Выкл отображения нумерации строк
d)Вкл/Выкл отображения панели инструментов
e)Вкл/Выкл отображения строки состояния
f)Вкл/Выкл подсветки синтаксиса
g)Выбор синтаксиса (для подсветки)[дочернее меню] -доступно всегда, один синтаксис в дочернем меню выбран всегда
h)Выбор/Редактирование стиля подсветки [дочернее меню] -для текущего синтаксиса, по умолчанию выбран Default
i)Изменить [кнопка] -измененный стиль сохраняется в файл, имя файла становится именем стиля, стиль становится активным

ii)Загрузка стиля из файла [кнопка] -имя файла становится именем стиля, стиль становится активным
iii)Обязательная кнопка Default
iv)доступные стили[перечисляются все стили которые были обнаружены]
5)Справка
а)О программе -открывает модальное окно содержащее фото и имя автора, дату сборки, версию Qt с которой собиралось, версию Qt с которой запущено, кнопку закрывающую окно
Ниже главного меню располагается панель инструментов(отображение которойконтролируется в меню Вид)с кнопками (с картинками, текстовое описание во всплывающей подсказке):
1)Новый документ
2)Открыть
3)Сохранить
4)Отменить
5)Повторить
6)Копировать
7)Вырезать
8)Вставить
9)Найти / Найти и заменить (как выпадающая кнопка) -открывающая (немодальное)диалоговое окно
В центральной части окна располагается область для редактирования текста. При нажатии левой кнопки курсор вставляется в позицию. При двойном нажатии левой кнопки выделяется слово под курсором. При нажатии правой кнопки(далее -если нет=*, есть=** выделения)курсор вставляется в позицию и выдается контекстное меню (кнопки могут быть неактивны): отменить, повторить, выделить*, выделить строку*, копировать**, вырезать**, вставить (** или если есть текст в буфере обмена), удалить**, выделить все.

Нижнюю часть окна занимает строка состояния. Информация разделена на три столбца: текущая позиция курсора (строка:столбец); время (и дата если другие сутки) последней операции (сохранения/изменения); количество строк, слов, символов, размер в килобайтах.
Дополнительный балл –подсветка синтаксиса Си++11/14 (Си++17, Си++20)
Дополнительный балл –сохранение настроек приложения в ini-файл.

1 Общий алгоритм решения поставленной задачи

1.2 Классы

Для решения поставленной задачи были разработаны пользовательские классы Info, Find и FindAndReplac наследуемые от класса QDialog, используемые для вывода модального окна и считывания пользовательского ввода с клавиатуры; MainWindow, наследуемый от QMainWindow, используемый для вывода на экран основного окна и всех компонентов программы;TextEdit, наследуемые от класса QTextEdit, в методах которого происходит создание текстового редактора; Highlighter, наследуемый от QSyntaxHighlighter, используемый для подсветки синтаксиса (C,C++98/03,C++11/14)

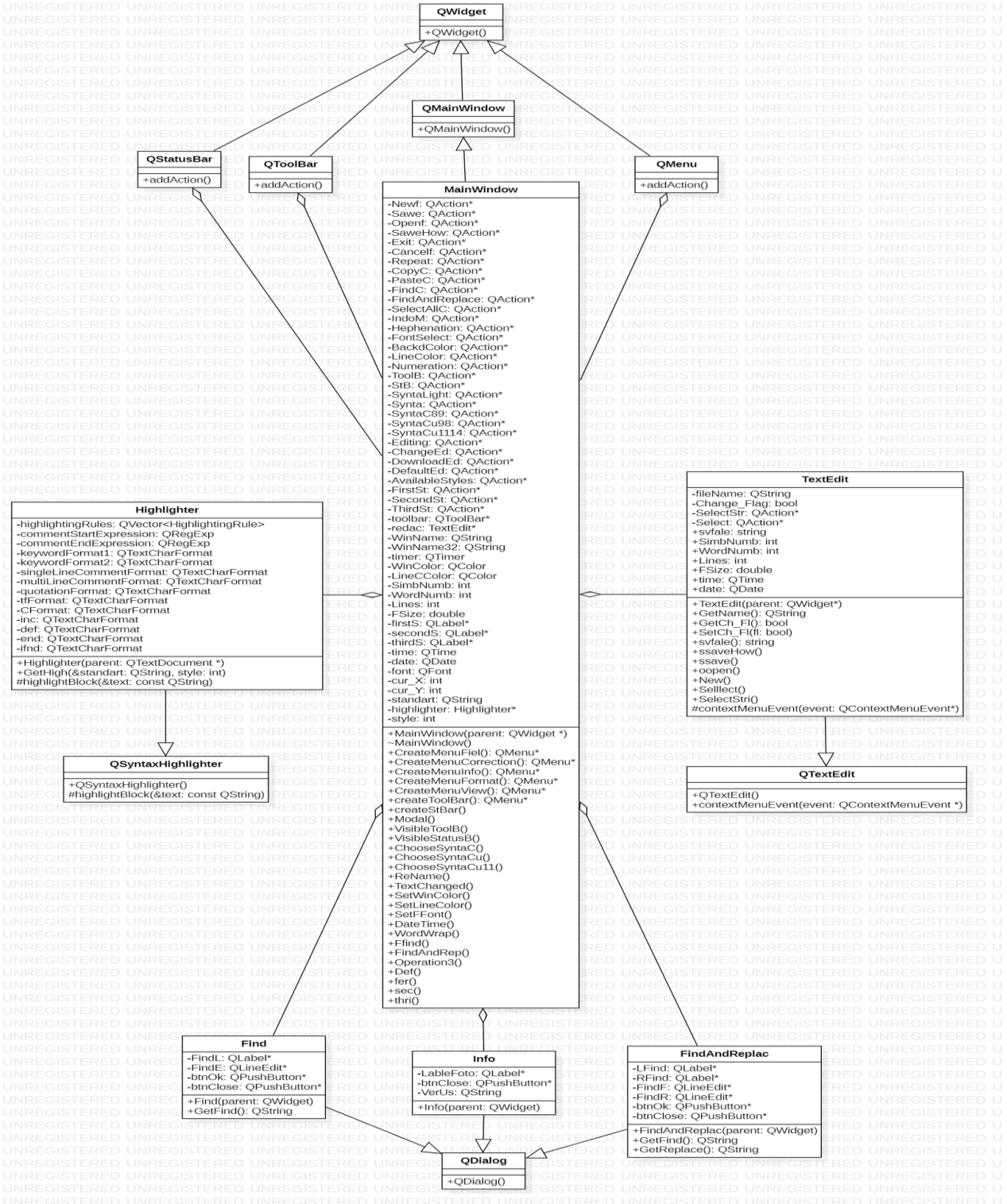


Рисунок 1 – UML 2.0 диаграмма классов

1.2 Поля классов

1.2.1 Класс MainWindow

QAction* Newf
Кнопка для создания нового файла
QAction* Sawe
Кнопка для сохранения файла
QAction* Openf
Кнопка для вызова модального окна открытия файлов
QAction* SaweHow
Кнопка для сохранения файлов как
QAction* Exit
Кнопка для выхода
QAction* Cancelf
Кнопка отмены
QAction* Repeat
Кнопка повтора
QAction* CopyC
Кнопка копирования
QAction* CutC
Кнопка вырезания
QAction* PasteC
Кнопка вставки
QAction* FindC
Кнопка вызывающая модальное окно, которое ищет слово
QAction* FindAndReplace
Кнопка вызывающая модальное окно, которое ищет и заменяет слово
QAction* SelectAllC
Кнопка выделяющая весь текст
QAction* InfoM
Кнопка вызывающая модальное окно, содержащее информацию о программе
QAction* Hyphenation
Залипающая кнопка ответственная за перенос текста по словам
QAction* FontSelect
Кнопка открывающая модальное окно выбора шрифта
QAction* BackdColor
Кнопка открывающая модальное окно выбора цвета задника
QAction* LineColor
Кнопка открывающая модальное окно выбора цвета строки
QAction* ToolB
Залипающая кнопка ответственная за видимость тулбара
QAction* StB
Залипающая кнопка ответственная за видимость статусбара
QAction* SyntaLight
Кнопка отвечающая за включение подсветки синтаксиса

QAction* Synta
Кнопка, открывающая дочернее меню, отвечающее за выбор языка
QAction* SyntaC89
Кнопка выбора языка C89
QAction* SyntaCu98
Кнопка выбора языка C98/03
QAction* SyntaCu1114
Кнопка выбора языка C11/14
QAction* Editing
Кнопка, открывающее дочернее меню, отвечающее за стили подсветки
QAction* DefaultEd
Кнопка выбора стандартной подсветки
QAction* AvailableStyles
Кнопка, открывающее дочернее меню, отвечающее за выбор доступного стиля
QAction* FirstSt
Кнопка выбора первого стиля
QAction* SecondSt
Кнопка выбора второго стиля
QAction* ThirdSt
Кнопка выбора третьего стиля
QToolBar* toolbar
Тулбар главного окна
TextEdit* redac
Текстовый редактор, помещенный в центр главного окна
QString WinName
Имя файла
QString WinName32
Имя файла, ограниченное 32 символами
QTimer timer
Таймер, отвечающий за отслеживание изменений в файле
QColor WinColor
Цвет заднего фона
QColor LineCColor
Цвет строки текста
int SimbNumb
Количество символов
int WordNumb
Количество слов
int Lines
Количество линий
double FSize
Размер файла в килобайтах
QLabel* firstS
Первое поле статусбара

QLabel* seconds
Второе поле статусбара
QLabel* thirdS
Третье поле статус бара
QTime time
Время последнего изменения файла
QDate date
Дата последнего изменения файла
QFont font
Шрифт текста
int cur_X
Позиция текстового курсора
int cur_Y
Позиция текстового курсора
QString standard
Стандарт подсветки синтаксиса
Highlighter* highlighter
Объект класса подсветки текста
int style
Стиль подсветки

1.2.2 Класс Info
QLabel* LableFoto
Фото создателя
QPushButton* btnClose
Кнопка закрывающая модальное окно
QString VerUs
Пользовательская версия Qt

1.2.3 Класс TextEdit
QString filename
Содержит название файла
bool Change_Flag
Флаг изменения файла
QAction* SelectStr
Действие, выделяющее строку
QAction* Select
Действие, выделяющее слова под текстовым курсором
string svfale
Вспомогательное название файла
int SimbNumb
Количество символов
int WordNumb
Количество слов

int Lines
Количество строк
double FSize
Размер файла
QTime time
Время последнего изменения
QDate date
Дата последнего изменения

1.2.4 Класс Find
QLabel* FindL
Лейбл с текстом “find”
QLineEdit* FindE
Изменяемое поле, в которое вводят слово, которое хотят найти
QPushButton* btnOk
Кнопка Ок
QPushButton* btnClose
Кнопка закрытия

1.2.5 Класс FindAndReplac
QLabel* LFind
Лейбл с текстом “find”
QLabel* RFind
Лейбл с текстом “replace”
QLineEdit* FindF
Изменяемое поле, в которое вводят слово, которое хотят найти
QLineEdit* FindR
Изменяемое поле, в которое вводят слово, на которое хотят заменить найденное
QPushButton* btnOk
Кнопка Ок
QPushButton* btnClose
Кнопка закрытия

1.2.6 Класс Highlighter
struct HighlightingRule
Структура с правилами подсветки
QVector highlihgtingRules
Вектор со структурой
QRegExp commentStart(End)Expression
Постоянные выражения для начала(конца) комментариев
QTextCharFormat keywordFormat1-ifnd
Формат подсветки для различных групп символов

1.3Поля Классов

1.3.1 Поля класса MainWindow

MainWindow(parent: QWidget *)
Конструктор класса
~MainWindow()
Деструктор класса
CreateMenuFiel():QMenu*
Метод, который создает и возвращает меню
CreateMenuCorrection():QMenu*
Метод, который создает и возвращает меню
CreateMenuInfo():QMenu*
Метод, который создает и возвращает меню
CreateMenuFormat():QMenu*
Метод, который создает и возвращает меню
CreateMenuView():QMenu*
Метод, который создает и возвращает меню
createToolBar():QToolBar*
Метод, который создает и возвращает тул бар
createStBar()
Метод, который создает статус бар
Modal()
Слот вызывающий модальное окно с информацией о программе
VisibleToolB()
Слот, который скрывает/открывает тул бар
VisibleStatusB()
Слот, который скрывает/открывает статус бар
ChooseSyntaC()
Слот, выбирающий язык подсветки C89
ChooseSyntaCu()
Слот, выбирающий язык подсветки C99/03
ChooseSyntaCu11()
Слот, выбирающий язык подсветки C11/14
ReName()
Слот, добавляющий в начало имени файла “” при изменении файла*
TextChanged()
Слот, изменяющий флаги изменения текста при изменении текста
SetWinColor()
Слот устанавливающий цвет, выбранный пользователем, заднего фона
SetLineColor()
Слот устанавливающий цвет, выбранный пользователем, строки
SetFFont()
Слот устанавливающий шрифт, выбранный пользователем
DateTime()

Слот устанавливающий в статус бар время и дату при необходимости
WordWrap()
Слот отвечающий за перенос по словам
Ffind()
Слот, вызывающий модальное окно, благодаря которому ищется слово
FindAndRep()
Слот, вызывающий модальное окно, благодаря которому ищется и заменяется слово
Def()
Слот ставящий "дефолтный" стиль подсветки
fer()
Слот ставящий первый стиль подсветки
sec()
Слот ставящий второй стиль подсветки
thri()
Слот ставящий третий стиль подсветки
pods()
Слот включающий и отключающий подсветку

1.3.2 Класс Info
Info(QWidget *parent = 0)
Конструктор класса

1.3.3 Класс TextEdit
TextEdit(QWidget *parent = 0)
Конструктор класса
QString GetName():QString
Метод возвращающий название файла
contextMenuEvent(event: QContextMenuEvent *)
Переопределенный метод класса родителя, служащий для вызова контекстного меню
ssaveHow()
Слот, который сохраняет как
ssave()
Слот сохраняющий файл
oopen()
Слот, открывающий файл
New()
Слот создающий новый файл
Selllect()
Слот, выбирающий слово под текстовым курсором
SelectStri()
Слот, выбирающий строку

1.3.3Класс Find

Find(QWidget *parent = 0)

Конструктор класса

GetFind():QString

Метод возвращающий текст поисковой строки

1.3.4 Класс FindAndReplac

FindAndReplac(parent: QWidget *)

Конструктор класса

GetFind():QString

Метод возвращающий текст поисковой строки

GetReplace():QString

Метод возвращающий текст строки замены

1.1.1 Класс Highlighter

Highlighter(parent: QTextDocument *)

Конструктором класса

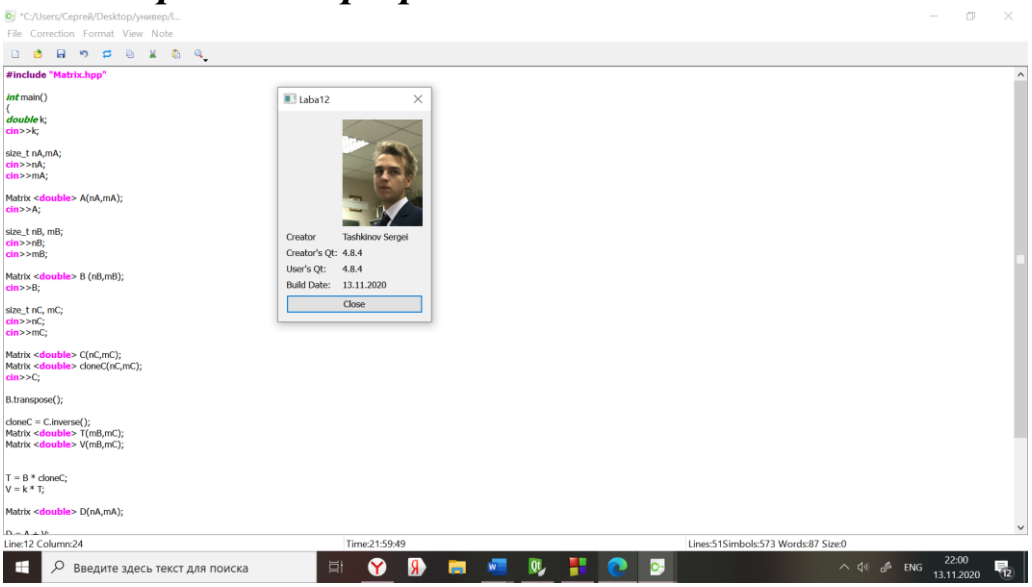
getHigh(&standard: QString, style:int)

Устанавливает новые значения параметров подсветки.

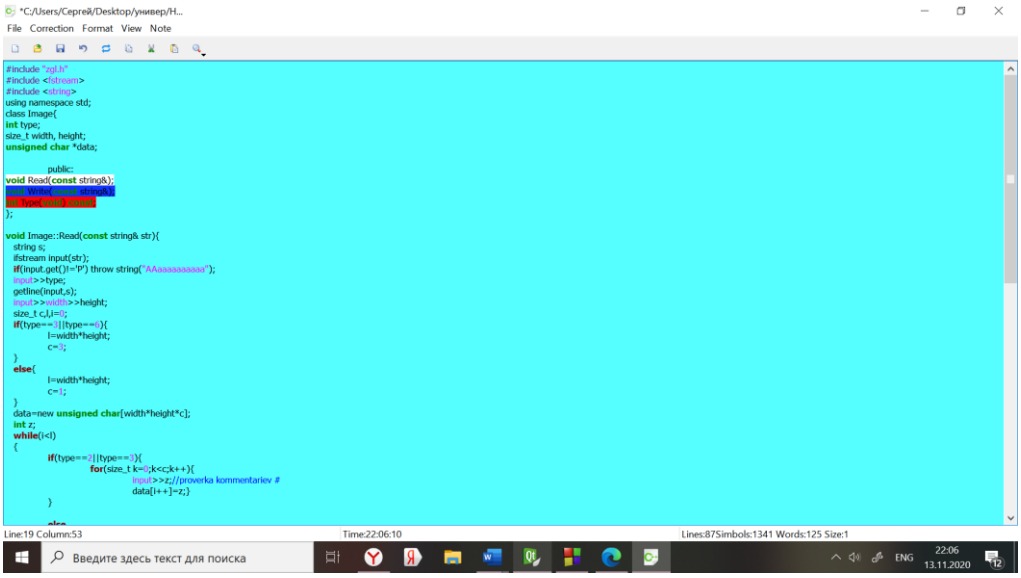
highlightBlock(&text: const QString)

Переопределенный метод класса-родителя, в котором происходит отрисовка подсветки.

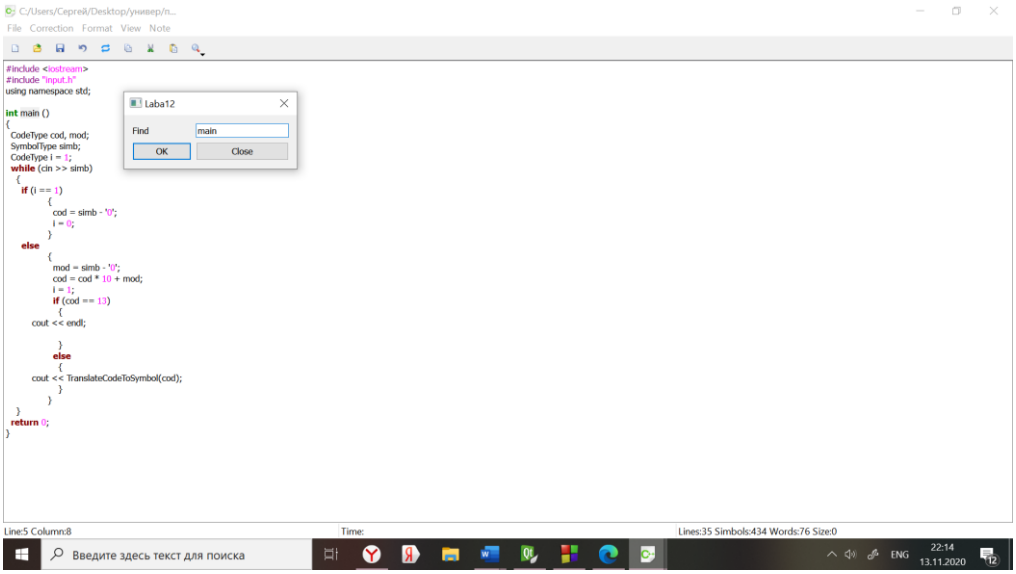
2.Тестирование программы



Листинг 1



Листинг 2



Листинг 3

3.*Приложение А*

A1 main.cpp

```
#include <QApplication>
#include "mainwindow.h"

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

A2 mainwindow.h

```
#ifndef MAINWINDOW_H
#define MAINWINDOW_H

#include <info.h>
#include <textedit.h>
#include <find.h>
#include <findandreplace.h>
#include <texthigligt.h>

#include <QMainWindow>
#include <QMenu>
#include <QMenuBar>
#include <QAction>
#include <QWidget>
#include <QApplication>
#include <QToolBar>
#include <QStatusBar>
#include <QTimer>
#include <QFont>
#include <QColor>
#include <QColorDialog>
#include <QFontDialog>
#include <QPalette>
#include <QPainter>

class MainWindow : public QMainWindow
{
    Q_OBJECT
private:
QAction* Newf;
QAction* Sawe;
QAction* Openf;
QAction* SaweHow;
QAction* Exit;
QAction* Cancelf;
QAction* Repeat;
QAction* CopyC ;
QAction* CutC;
QAction* PasteC;
```

```
QAction* FindC ;
QAction* FindAndReplace;
QAction* SelectAllC;
QAction* InfoM;
QAction* Hyphenation;
QAction* FontSelect;
QAction* BackdColor;
QAction* LineColor;
QAction* Numeration;
QAction* ToolB;
QAction* StB;
QAction* SyntaLight;
QAction* Synta;
QAction* SyntaC89;
QAction* SyntaCu98;
QAction* SyntaCu1114;
QAction* Editing;
QAction* ChangeEd;
QAction* DownloadEd;
QAction* DefaultEd;
QAction* AvailableStyles;
QAction* FirstSt;
QAction* SecondSt;
QAction* ThirdSt;
QToolBar* toolbar;
TextEdit* redac;
QString WinName;
QString WinName32;
QTimer timer;
QColor WinColor;
QColor LineCColor;
int SimbNumb;
int WordNumb;
int Lines;
double FSize;
QLabel* firstS;
QLabel* secondS;
QLabel* thirdS;
QTime time;
QDate date;
QFont font;
int cur_X;
int cur_Y;
QString standart;
Highlighter* highlighter;
int style;
public:
    MainWindow(QWidget *parent = 0);
    ~MainWindow();
    QMenu* CreateMenuFiel();
    QMenu* CreateMenuCorrection();
    QMenu* CreateMenuInfo();
    QMenu* CreateMenuFormat();
    QMenu* CreateMenuView();
    QToolBar* createToolBar();
    void createStBar();
public slots:
    void Modal();
```

```
    void VisibleToolB();
    void VisibleStatusB();
    void ChooseSyntaC();
    void ChooseSyntaCu();
    void ChooseSyntaCu11();
    void ReName();
    void TextChanged();
    void SetWinColor();
    void SetLineColor();
    void SetFFont();
    void DateTime();
    void WordWrap();
    void Ffind();
    void FindAndRep();
    void Def();
    void fer();
    void sec();
    void thri();
    void pods();
};

#endif // MAINWINDOW_H
```

A3 mainwindow.cpp

```
#include "mainwindow.h"

MainWindow::MainWindow(QWidget *parent)
    : QMainWindow(parent),standart("C89"),style(0)
{
    this->resize(1000,800);
    QPalette WindowPal(palette());
    WindowPal.setColor(QPalette::Background, Qt::white);
    this->setPalette(WindowPal);
    QPixmap cpix("iconC.png");
    this->setWindowIcon(QIcon(cpix));
    this->redac=new TextEdit();

    setCentralWidget(redac);
    CreateMenuFiel();
    CreateMenuCorrection();
    CreateMenuFormat();
    CreateMenuView();
    CreateMenuInfo();
    addToolBar(Qt::TopToolBarArea, createToolBar());
    createStBar();
    connect(redac,SIGNAL(textChanged()),this,SLOT(TextChanged()));
    connect(&timer,SIGNAL(timeout()),this,SLOT(ReName()));
    connect(redac,SIGNAL(textChanged()),this,SLOT(DateTime()));
    highlighter = new Highlighter(redac->document());
    highlighter->getHigh(standart, style);
    timer.start(10);
}
QMenu* MainWindow::CreateMenuFiel()
{
QMenu* File;
File =menuBar()->addMenu("&File");
```

```
this->Newf= new  QAction("&New",this);
File->addAction(Newf);
connect(Newf,SIGNAL(triggered()),redac,SLOT(New()));
this->Openf= new QAction("&Open",this);
File->addAction(Openf);
connect(Openf,SIGNAL(triggered()),redac,SLOT(oopen()));
this->Sawe= new QAction("&Save",this);
File->addAction(Sawe);
connect(Sawe,SIGNAL(triggered()),redac,SLOT(ssave()));
this->SaweHow= new QAction("&Save How",this);
File->addAction(SaweHow);
connect(SaweHow,SIGNAL(triggered()),redac,SLOT(ssaveHow()));
File->addSeparator();
this->Exit= new QAction("&Exit",this);
File->addAction(Exit);
connect(Exit,SIGNAL(triggered()), qApp, SLOT(quit()));
return File;
}
QMenu* MainWindow::CreateMenuCorrection()
{
 QMenu* Correction;
 Correction=menuBar()->addMenu("&Correction");
 this->Cancelf= new  QAction("&Cancel",this);
 Correction->addAction(Cancelf);
 connect(Cancelf,SIGNAL(triggered()),redac,SLOT(undo()));
 this->Repeat= new QAction("&Repeat",this);
 Correction->addAction(Repeat);
 connect(Repeat,SIGNAL(triggered()),redac,SLOT(redo()));
 this->CopyC= new QAction("&Copy",this);
 Correction->addAction(CopyC);
 connect(CopyC,SIGNAL(triggered()),redac,SLOT(copy()));
 this->CutC= new QAction("&Cut",this);
 Correction->addAction(CutC);
 connect(CutC,SIGNAL(triggered()),redac,SLOT(cut()));
 this->PasteC= new QAction("&Paste",this);
 Correction->addAction(PasteC);
 connect(PasteC,SIGNAL(triggered()),redac,SLOT(paste()));
 this->FindC= new QAction("&Find",this);
 Correction->addAction(FindC);
 connect(FindC,SIGNAL(triggered()),this,SLOT(Ffind()));
 this->FindAndReplace = new QAction("&Find and Replace",this);
 Correction->addAction(FindAndReplace);
 connect(FindAndReplace,SIGNAL(triggered()),this,SLOT(FindAndRep()));
 this->SelectAllC= new QAction("&Select All",this);
 Correction->addAction(SelectAllC);
 connect(SelectAllC,SIGNAL(triggered()),redac,SLOT(selectAll()));
 return Correction;
}

QMenu* MainWindow::CreateMenuInfo()
{
  QMenu* Note;
  Note=menuBar()->addMenu("&Note");
  this->InfoM= new  QAction("&Info",this);
  Note->addAction(InfoM);
  connect(InfoM,SIGNAL(triggered()),this,SLOT( Modal()));
  return Note;
}
```

```
QToolBar* MainWindow::createToolBar()
{
    this->toolbar = addToolBar("main toolbar");
    toolbar->setMovable(false);
    QPalette toolBarPal(palette());
    toolBarPal.setColor(QPalette::Background, Qt::gray);
    toolbar->setAutoFillBackground(true);
    toolbar->setPalette(toolBarPal);
    QPixmap newpix("new.png");
    QPixmap openpix("open.png");
    QPixmap savepix("save.png");
    QPixmap canpix("cancel.png");
    QPixmap repeatpix("repeat.png");
    QPixmap copypix("copy.png");
    QPixmap cutpix("cut.png");
    QPixmap findpix("find.png");
    QPixmap findAndpix("findnext.png");
    QPixmap pastepix("insert.png");
    this->Newf=toolbar->addAction(QIcon(newpix),"New");
    connect(Newf,SIGNAL(triggered()),redac,SLOT(New()));
    this->Openf=toolbar->addAction(QIcon(openpix), "Open File");
    connect(Openf,SIGNAL(triggered()),redac,SLOT(oopen()));
    this->Sawe=toolbar->addAction(QIcon(savepix), "Save");
    connect(Sawe,SIGNAL(triggered()),redac,SLOT(ssave()));
    this->Cancelf=toolbar->addAction(QIcon(canpix), "Cancel");
    connect(Cancelf,SIGNAL(triggered()),redac,SLOT(undo()));
    this->Repeat=toolbar->addAction(QIcon(repeatpix), "Repeat");
    connect(Repeat,SIGNAL(triggered()),redac,SLOT(redo()));
    this->CopyC=toolbar->addAction(QIcon(copypix), "Copy");
    connect(CopyC,SIGNAL(triggered()),redac,SLOT(copy()));
    this->CutC=toolbar->addAction(QIcon(cutpix), "Cut");
    connect(CutC,SIGNAL(triggered()),redac,SLOT(cut()));
    this->PasteC=toolbar->addAction(QIcon(pastepix),"Past");
    connect(PasteC,SIGNAL(triggered()),redac,SLOT(paste()));
    this->FindC=toolbar->addAction(QIcon(findpix), "Find");
    connect(FindC,SIGNAL(triggered()),this,SLOT(Ffind()));

    QMenu * MenFind = new QMenu(this);
    this->FindAndReplace=MenFind->addAction(QIcon(findAndpix), "Find
and Replace");

connect(FindAndReplace,SIGNAL(triggered()),this,SLOT(FindAndRep()));
    this->FindC->setMenu(MenFind);
    return toolbar;
}
QMenu* MainWindow::CreateMenuFormat()
{
    QMenu* Format;
    Format=menuBar()->addMenu("&Format");
    this->Hyphenation= new  QAction("&Word wrap",this);
    Hyphenation->setCheckable(true);
    Hyphenation->setChecked(true);
    Format->addAction(Hyphenation);
    connect(Hyphenation,SIGNAL(triggered()),this,SLOT(WordWrap()));
    this->FontSelect= new QAction("&Font Selection",this);
    Format->addAction(FontSelect);
    connect(FontSelect,SIGNAL(triggered()),this,SLOT(SetFFont()));
```

```
    return Format;
}
QMenu* MainWindow::CreateMenuView()
{
    QMenu* View;
    View=menuBar()->addMenu("&View");
    this->BackdColor= new  QAction("Backgraund Color",this);
    View->addAction(BackdColor);
    connect(BackdColor,SIGNAL(triggered()),this,SLOT(SetWinColor()));
    this->LineColor= new  QAction("Line Color",this);
    View->addAction(LineColor);
    connect(LineColor,SIGNAL(triggered()),this,SLOT(SetLineColor()));
    this->Numeration= new  QAction("Numeration",this);
    Numeration->setChecked(false);
    Numeration->setDisabled(true);
    View->addAction(Numeration);
    this->ToolB= new  QAction("ToolBar visible",this);
    ToolB->setCheckable(true);
    ToolB->setChecked(true);
    connect(ToolB,SIGNAL(triggered()),this,SLOT(VisibleToolB()));
    View->addAction(ToolB);
    this->StB= new  QAction("StatusBar visible",this);
    StB->setCheckable(true);
    StB->setChecked(true);
    connect(StB,SIGNAL(triggered()),this,SLOT(VisibleStatusB()));
    View->addAction(StB);
    this->SyntaLight = new QAction("Syntax Backlight",this);
    SyntaLight->setCheckable(true);
    SyntaLight->setChecked(true);
    View->addAction(SyntaLight);
    connect(SyntaLight,SIGNAL(triggered()),this,SLOT(pods()));
    this->Synta = new QAction("Syntax",this);
    View->addAction(Synta);

    QMenu* MenSynta = new QMenu(this);
    this->SyntaC89=MenSynta->addAction("C89");
    SyntaC89->setCheckable(true);
    SyntaC89->setChecked(true);
    connect(SyntaC89,SIGNAL(triggered()),this,SLOT(ChooseSyntaC()));
    this->SyntaCu98=MenSynta->addAction("C++98/03");
    SyntaCu98->setCheckable(true);
    connect(SyntaCu98,SIGNAL(triggered()),this,SLOT(ChooseSyntaCu()));
    this->Synta->setMenu(MenSynta);
    this->SyntaCu1114=MenSynta->addAction("C++11/14");
    SyntaCu1114->setCheckable(true);

connect(SyntaCu1114,SIGNAL(triggered()),this,SLOT(ChooseSyntaCu11()));


    this->Editing = new QAction("Editing",this);
    View->addAction(Editing);

    QMenu* MenEd = new QMenu(this);
    this->ChangeEd=MenEd->addAction("Change");
    ChangeEd->setDisabled(true);
    this->DownloadEd=MenEd->addAction("Downlaud Style from file");
    DownloadEd->setDisabled(true);
    this->DefaultEd=MenEd->addAction("Default");
```

```
    connect(DefaultEd,SIGNAL(triggered()),this,SLOT(Def()));
    this->AvailableStyles=MenEd->addAction("Available Styles");
    QMenu* AvailableStyl= new  QMenu(this);  //MenEd-
>addAction("Available Styles");
    this->FirstSt=AvailableStyl->addAction("First Style");
    connect(FirstSt,SIGNAL(triggered()),this,SLOT(fer()));
    this->SecondSt=AvailableStyl->addAction("Second Style");
    connect(SecondSt,SIGNAL(triggered()),this,SLOT(sec()));
    this->ThirdSt=AvailableStyl->addAction("Third Style");
    connect(SecondSt,SIGNAL(triggered()),this,SLOT(thri()));


    this->AvailableStyles->setMenu(AvailableStyl);
    this->Editing->setMenu(MenEd);

    return View;
}

void MainWindow::createStBar()
{
    this->firstS= new QLabel(this);
    this->secondS= new QLabel(this);
    this->thirdS= new QLabel("",this);
    statusBar()->addWidget(firstS,1);
    statusBar()->addWidget(secondS,1);
    statusBar()->addWidget(thirdS,1);
}

void MainWindow::Modal()
{
    Info* dialog = new Info();
    dialog->exec();
    delete dialog;
}
void MainWindow::VisibleToolB()
{
 if (this->ToolB->isChecked())
    {
        this->toolbar->show();
    }
 else
    {
        this->toolbar->hide();
    }
}

void MainWindow::VisibleStatusB()
{
    if(this->StB->isChecked())
      {
        statusBar()->show();
      }
    else
      {
       statusBar()->hide();
      }
}
```

```
void MainWindow::ChooseSyntaC()
{
    if (this->SyntaC89->isChecked())
    {
       this->SyntaCu98->setChecked(false);
       this->SyntaCu1114->setChecked(false);
       standart = "C89";
            delete highlighter;
            highlighter = new Highlighter(redac->document());
            highlighter->getHigh(standart, style);

    }
    else
     {
       this->SyntaCu98->setChecked(true);
       standart = "C++99/03";
             delete highlighter;
             highlighter = new Highlighter(redac->document());
             highlighter->getHigh(standart, style);

     }
}
void MainWindow::ChooseSyntaCu()
{
   if(this->SyntaCu98->isChecked())
   {
       this->SyntaC89->setChecked(false);
       this->SyntaCu1114->setChecked(false);
       standart = "C++99/03";
             delete highlighter;
             highlighter = new Highlighter(redac->document());
             highlighter->getHigh(standart, style);
   }
   else
   {
     this->SyntaCu1114->setChecked(true);
     standart = "C89";
            delete highlighter;
            highlighter = new Highlighter(redac->document());
            highlighter->getHigh(standart, style);
   }
}

void MainWindow::ChooseSyntaCu11()
{
    if(this->SyntaCu1114->isChecked())
     {
        this->SyntaC89->setChecked(false);
        this->SyntaCu98->setChecked(false);
        standart = "C++11/14";
        delete highlighter;
            highlighter = new Highlighter(redac->document());
            highlighter->getHigh(standart,style);
     }
    else
     {
        this->SyntaC89->setChecked(true);
        this->SyntaCu98->setChecked(false);
```

```
        standart = "C89";
               delete highlighter;
               highlighter = new Highlighter(redac->document());
               highlighter->getHigh(standart, style);
     }
}

void MainWindow::Def()
{
    style = 0;
    delete highlighter;
        highlighter = new Highlighter(redac->document());
        highlighter->getHigh(standart,style);
}

void MainWindow::fer()
{
    style = 1;
        delete highlighter;
        highlighter = new Highlighter(redac->document());
        highlighter->getHigh(standart,style);
}

void MainWindow::sec()
{
    style = 2;
        delete highlighter;
        highlighter = new Highlighter(redac->document());
        highlighter->getHigh(standart,style);

}
void MainWindow::thri()
{
    style = 3;
        delete highlighter;
        highlighter = new Highlighter(redac->document());
        highlighter->getHigh(standart,style);
}

void MainWindow::ReName()
{
    this->WinName=redac->GetName();
    if(WinName.size()<=32)
       {
        if(redac->GetCh_Fl())
         {
            this->WinName32="*"+WinName;
            this->setWindowTitle(WinName32);

            this->WordNumb = redac-
>toPlainText().split(QRegExp("(\\s|\\n|\\r)+")
               , QString::SkipEmptyParts).count();

            this->SimbNumb=redac->toPlainText().length();
            this->Lines =(redac->document()->lineCount());
            this->FSize=redac->FSize;
            this->thirdS->setText("Lines:" + QString::number(Lines)+"
"+"Simbols:"+QString::number(SimbNumb)
```

```
                                        +"
"+"Words:"+QString::number(WordNumb)+"
"+"Size:"+QString::number(FSize));

             QTextCursor control_cur=redac->textCursor();
             this->cur_X=control_cur.columnNumber();
             this->cur_Y = control_cur.blockNumber()+1;
             this->firstS->setText("Line:"+QString::number(cur_Y)+"
"+"Column:" + QString::number(cur_X));

          }
         else
          {
           this->setWindowTitle(WinName);

             this->WordNumb=redac->WordNumb;
             this->SimbNumb=redac->SimbNumb;
             this->Lines=redac->Lines;
             this->FSize=redac->FSize;
             this->thirdS->setText("Lines:" + QString::number(Lines)+"
"+"Simbols:"+QString::number(SimbNumb)
                                        +"
"+"Words:"+QString::number(WordNumb)+"
"+"Size:"+QString::number(FSize));

             this->date=redac->date;
             this->time=redac->time;
             if(this->date==QDate::currentDate())
               {
                this->secondS->setText("Time:"+ time.toString());
               }
             else
               {
                 this->secondS->setText("Date"+date.toString()+"
"+"Time:"+ time.toString());
                 this->date=QDate::currentDate();
               }
             QTextCursor control_cur=redac->textCursor();
             this->cur_X=control_cur.columnNumber();
             this->cur_Y = control_cur.blockNumber()+1;
             this->firstS->setText("Line:"+QString::number(cur_Y)+"
"+"Column:" + QString::number(cur_X));
          }
        }
     else
        {
         if(redac->GetCh_Fl())
          {
            this->WinName32=this->WinName;
            WinName32.resize(32);
            this->WinName32="*"+WinName32+"...";
            this->setWindowTitle(WinName32);

            this->WordNumb = redac-
>toPlainText().split(QRegExp("(\\s|\\n|\\r)+")
               , QString::SkipEmptyParts).count();

            this->SimbNumb=redac->toPlainText().length();
```

```
            this->Lines =(redac->document()->lineCount());
            this->FSize=redac->FSize;
            this->thirdS->setText("Lines:" +
QString::number(Lines)+""+"Simbols:"+QString::number(SimbNumb)
                                    +"
"+"Words:"+QString::number(WordNumb)+"
"+"Size:"+QString::number(FSize));

            QTextCursor control_cur=redac->textCursor();
            this->cur_X=control_cur.columnNumber();
            this->cur_Y = control_cur.blockNumber()+1;
            this->firstS->setText("Line:"+QString::number(cur_Y)+"
"+"Column:" + QString::number(cur_X));

          }
         else
          {
             this->WinName32=this->WinName;
             WinName32.resize(32);
             this->WinName32=WinName32+"...";
             this->setWindowTitle(WinName32);

             this->WordNumb=redac->WordNumb;
             this->SimbNumb=redac->SimbNumb;
             this->Lines=redac->Lines;
             this->FSize=redac->FSize;
             this->thirdS->setText("Lines:" + QString::number(Lines)+"
"+"Simbols:"+QString::number(SimbNumb)
                                     +"
"+"Words:"+QString::number(WordNumb)+"
"+"Size:"+QString::number(FSize));

             this->date=redac->date;
             this->time=redac->time;

             if(this->date==QDate::currentDate())
               {
                this->secondS->setText("Time:"+ time.toString());
               }
             else
               {
                 this->secondS->setText("Date"+date.toString()+"
"+"Time:"+ time.toString());
                 this->date=QDate::currentDate();
               }

             QTextCursor control_cur=redac->textCursor();
             this->cur_X=control_cur.columnNumber();
             this->cur_Y = control_cur.blockNumber()+1;
             this->firstS->setText("Line:"+QString::number(cur_Y)+"
"+"Column:" + QString::number(cur_X));
          }
        }
}
void MainWindow::SetWinColor()
{
  this->WinColor=QColorDialog::getColor(Qt::white,this);
  QPalette pallet;
```

```
    if(WinColor.isValid())
     {
        pallet.setColor(QPalette::Base,WinColor);
        qApp->setPalette(pallet);
     }
    else
    {
      WinColor=Qt::white;
      pallet.setColor(QPalette::Base,WinColor);
      qApp->setPalette(pallet);
    }
}

void MainWindow::SetLineColor()
{
    this->LineCColor=QColorDialog::getColor(Qt::white,this);
      if(LineCColor.isValid())
       {
        this->redac->setTextBackgroundColor(LineCColor);

       }
      else
      {
       LineCColor=Qt::white;
       this->redac->setTextBackgroundColor(LineCColor);
      }
}

void MainWindow::SetFFont()
{
    bool ok;
    this->font = QFontDialog::getFont(&ok, QFont("Courier New", 12),
this, "Choose Font");
    if (ok)
     {
       redac->setCurrentFont(font);
     }
    else
     {
      redac->setCurrentFont(QFont("Courier New", 12));
     }
}

void MainWindow::DateTime()
{
    this->time=QTime::currentTime();
    if(this->date==QDate::currentDate())
      {
       this->secondS->setText("Time:"+ time.toString());
      }
    else
      {
        this->secondS->setText("Date"+date.toString()+" "+"Time:"+
time.toString());
        this->date=QDate::currentDate();
      }

}
```

```
void MainWindow::WordWrap()
{
    if(this->Hyphenation->isChecked())
     {
        redac->setLineWrapMode(QTextEdit::FixedColumnWidth);
     }
    else
     {
        redac->setLineWrapMode(QTextEdit::NoWrap);
     }
}

void MainWindow::Ffind()
{
    Find* dialog = new Find();
    dialog->exec();
    redac->find(dialog->GetFind());
    delete dialog;
}

void MainWindow::pods()
{
    if(SyntaLight->isChecked())
     {
      this->SyntaC89->setEnabled(true);
      this->SyntaCu98->setEnabled(true);
      this->SyntaCu1114->setEnabled(true);
      this->DefaultEd->setEnabled(true);
      highlighter = new Highlighter(redac->document());
      highlighter->getHigh(standart,style);

     }
    else
    {
        this->SyntaC89->setEnabled(false);
        this->SyntaCu98->setEnabled(false);
        this->SyntaCu1114->setEnabled(false);
        this->DefaultEd->setEnabled(false);
        delete  highlighter;
    }


}

void MainWindow::FindAndRep()
{
    FindAndReplac* dialog=new FindAndReplac();
    dialog->exec();
    while (!redac->textCursor().isNull() && !redac-
>textCursor().atEnd())
    {
    redac->find(dialog->GetFind());
    redac->insertPlainText(dialog->GetReplace());
    redac->textCursor().movePosition(QTextCursor::EndOfWord);
    }
    delete dialog;
}
```

```
void MainWindow::TextChanged()
{
    bool j=true;
    redac->SetCh_Fl(j);
}

MainWindow::~MainWindow()
{
}
```

A4 textedit.h

```
#ifndef TEXTEDIT_H
#define TEXTEDIT_H

#include <string>
#include <QTextEdit>
#include <QFileDialog>
#include <QMessageBox>
#include <QFile>
#include <QTextStream>
#include <QTime>
#include <QDate>
#include <QTextCursor>
#include <QRect>
#include <QMenu>
#include <QAction>
#include <QList>
#include <QMouseEvent>

class TextEdit : public QTextEdit
{
    Q_OBJECT
private:
QString fileName;
bool Change_Flag;
QAction* SelectStr;
QAction* Select;
public:
    TextEdit(QWidget *parent = 0);
    QString GetName();
    bool GetCh_Fl();
    void SetCh_Fl(bool fl);
    std::string svfale;
    int SimbNumb;
    int WordNumb;
    int Lines;
    double FSize;
    QTime time;
    QDate date;
protected:
void  contextMenuEvent(QContextMenuEvent *event);
public slots:
    void ssaveHow();
    void ssave();
    void oopen();
```

```
    void New();
    void Selllect();
    void SelectStri();
};

#endif // TEXTEDIT_H
```

A5 textedit.cpp

```
#include "textedit.h"

TextEdit::TextEdit(QWidget *parent) :

QTextEdit(parent),fileName(""),Change_Flag(false),SimbNumb(0),WordNumb
(0),Lines(0),FSize(0)
  ,date(QDate::currentDate())
{
}
void TextEdit::oopen()
{
    this->fileName = QFileDialog::getOpenFileName(this, tr("Open
File"), "",
        tr("Text Files (*.txt);;C++ Files (*.cpp *.h);;C89 Files (*.c
*.h)"));

    if (fileName != "")
    {
        QFile file(fileName);
        if (!file.open(QIODevice::ReadOnly))
         {
            QMessageBox::critical(this, tr("Error"), tr("Could not
open file"));
            return;
         }
        QTextStream in(&file);
        this->setText(in.readAll());
        this->FSize=(file.size())/1000;
        file.close();
        this->Change_Flag=false;
    }
    this->WordNumb = this-
>toPlainText().split(QRegExp("(\\s|\\n|\\r)+")
       , QString::SkipEmptyParts).count();

    this->SimbNumb=this->toPlainText().length();
    this->Lines =(this->document()->lineCount());

}
void TextEdit::ssave()
{
    if (fileName != "")
    {

        svfale=fileName.toStdString();
        svfale=svfale.substr(fileName.size()-2,2);
        if(svfale=="xt"||svfale==".c"||svfale=="pp"||svfale==".h")
         {
```

```
            fileName=fileName;
          }
         else
          {
             fileName=fileName+".txt";
          }
         QFile file(fileName);
         if (!file.open(QIODevice::WriteOnly))
           {
             QMessageBox::critical(this, tr("Error"), tr("Could not
save file"));
             return;
           }
         else
         {
             QTextStream stream(&file);
             stream <<this->toPlainText();
             stream.flush();
             this->FSize=(file.size())/1000;
             file.close();
             this->Change_Flag=false;
         }
     }
     this->WordNumb = this-
>toPlainText().split(QRegExp("(\\s|\\n|\\r)+")
        , QString::SkipEmptyParts).count();

     this->SimbNumb=this->toPlainText().length();
     this->Lines =(this->document()->lineCount());
     this->time=QTime::currentTime();
     this->date=QDate::currentDate();

}

void TextEdit::ssaveHow()
{
     this->fileName = QFileDialog::getSaveFileName(this, tr("Save
File"), "",
     tr("Text Files (*.txt);;C++ Files (*.cpp *.h);;C89 Files (*.c
*.h)"));

     if (fileName != "")
     {
         QFile file(fileName);
         if (!file.open(QIODevice::WriteOnly))
           {
             QMessageBox::critical(this, tr("Error"), tr("Could not
save file"));
             return;
           }
         else
         {
             QTextStream stream(&file);
             stream <<this->toPlainText();
             stream.flush();
             this->FSize=(file.size())/1000;
             file.close();
             this->Change_Flag=false;
```

```
        }
    }
    this->WordNumb = this-
>toPlainText().split(QRegExp("(\\s|\\n|\\r)+")
       , QString::SkipEmptyParts).count();

    this->SimbNumb=this->toPlainText().length();
    this->Lines =(this->document()->lineCount());
    this->time=QTime::currentTime();
    this->date=QDate::currentDate();
}
void TextEdit::New()
{
  this->fileName="New";
  this->clear();
  this->Change_Flag=true;
  this->FSize=0;
}

void TextEdit::contextMenuEvent(QContextMenuEvent *event)
{


    QMenu *menu = createStandardContextMenu();
    this->SelectStr=menu->addAction("Select String");
    connect(SelectStr,SIGNAL(triggered()),this,SLOT(SelectStri()));
    this->Select=menu->addAction("Select");
    connect(Select,SIGNAL(triggered()),this,SLOT(Selllect()));
    menu->exec(event->globalPos());
    delete menu;
}

QString TextEdit::GetName()
{
  return this->fileName;
}
bool TextEdit::GetCh_Fl()
{
  return this->Change_Flag;
}
void TextEdit::SetCh_Fl(bool fl)
{
  this->Change_Flag=fl;
}
void TextEdit::Selllect()
{
    QTextCursor textCursor = this->textCursor();
    textCursor.select(QTextCursor::WordUnderCursor);
    this->setTextCursor(textCursor);
}
void TextEdit::SelectStri()
{
    QTextCursor cursor=this->textCursor();
    cursor.movePosition(QTextCursor::StartOfLine);
    cursor.movePosition(QTextCursor::EndOfLine,
QTextCursor::KeepAnchor);
    this->setTextCursor(cursor);
}
```

A6 info.h

```
#ifndef INFO_H
#define INFO_H

#include <QDialog>
#include <QLabel>
#include <QtGui>
#include <QPixmap>
#include <QBoxLayout>
#include <QtGlobal>
#include <QString>
#include <QIcon>
#include <QWidget>
class Info : public QDialog
{
    Q_OBJECT
private:
 QLabel* LableFoto;
 QPushButton* btnClose;
 QString VerUs;
public:
    Info(QWidget *parent = 0);
};

#endif // INFO_H
```

A7 info.cpp

```
#include "info.h"

Info::Info(QWidget *parent) :
    QDialog(parent, Qt::WindowTitleHint |
Qt::WindowSystemMenuHint),VerUs(qVersion())
{
    LableFoto= new QLabel;
    QLabel* labelCreator = new QLabel("Creator");
    QLabel* labelName = new QLabel("Tashkinov Sergei");
    QLabel* Date1 = new QLabel("Build Date:");
    QLabel* Date2 = new QLabel("13.11.2020");
    QLabel* labelVersiaCr = new QLabel("Creator's Qt:");
    QLabel* labelVersiaUS = new QLabel("User's Qt:");
    QLabel* labelVersiaCr1 = new QLabel("4.8.4");
    QLabel* labelVersiaUS1 = new QLabel(VerUs);
    this->btnClose = new QPushButton("Close");
    btnClose->setCheckable(true);
    connect(btnClose,SIGNAL(clicked()),SLOT(close()));
    QIcon Foto=QIcon::fromTheme("Foto",QIcon("Ffoto.jpg"));
    QPixmap pixmap =Foto.pixmap(QSize(150,200));
    LableFoto->setPixmap(pixmap);
    QGridLayout* layout = new QGridLayout;
    layout->addWidget(LableFoto,1,1);
    layout->addWidget(labelCreator, 2, 0);
    layout->addWidget(labelName, 2, 1);
    layout->addWidget(labelVersiaCr, 3, 0);
```

```
    layout->addWidget(labelVersiaCr1,3, 1);
    layout->addWidget(labelVersiaUS, 4, 0);
    layout->addWidget(labelVersiaUS1,4, 1);
    layout->addWidget(Date1,5,0);
    layout->addWidget(Date2,5,1);
    layout->addWidget(btnClose, 6,0,6,2);
    setLayout(layout);
}
```

A8 Find.h

```
#ifndef FIND_H
#define FIND_H

#include <QDialog>
#include <QLabel>
#include <QBoxLayout>
#include <QString>
#include <QWidget>
#include <QLineEdit>
#include <QPushButton>
class Find : public QDialog
{
    Q_OBJECT
private:
    QLabel* FindL;
    QLineEdit* FindE;
    QPushButton* btnOk;
    QPushButton* btnClose;
public:
   Find(QWidget *parent = 0);
   QString GetFind();
};

#endif // FIND_H
```

A9 find.cpp

```
#include "find.h"

Find::Find(QWidget *parent) :
    QDialog(parent, Qt::WindowTitleHint | Qt::WindowSystemMenuHint)
{
  FindE = new QLineEdit;
  FindL = new QLabel("Find");

  btnOk = new QPushButton("OK");
  btnOk->setCheckable(true);

  btnClose = new QPushButton("Close");
  btnClose->setCheckable(true);

  connect(btnOk, SIGNAL(clicked()), SLOT(accept()));
  connect(btnClose, SIGNAL(clicked()), SLOT(reject()));

  QGridLayout* layout = new QGridLayout;
```

```
  layout->addWidget(FindL, 0, 0);
  layout->addWidget(FindE, 0, 1);
  layout->addWidget(btnOk, 1, 0);
  layout->addWidget(btnClose, 1, 1);
  setLayout(layout);

}

QString Find::GetFind()
{
 return (FindE->text());
}
```

A10 findandreplace.h

```
#ifndef FINDANDREPLACE_H
#define FINDANDREPLACE_H

#include <QDialog>
#include <QLabel>
#include <QBoxLayout>
#include <QString>
#include <QWidget>
#include <QLineEdit>
#include <QPushButton>
class FindAndReplac : public QDialog
{
    Q_OBJECT
private:
    QLabel* LFind;
    QLabel* RFind;
    QLineEdit* FindF;
    QLineEdit* FindR;
    QPushButton* btnOk;
    QPushButton* btnClose;
public:
   FindAndReplac(QWidget *parent = 0);
   QString GetFind();
   QString GetReplace();

public slots:

};

#endif // FINDANDREPLACE_H
```

A11 findandreplace.cpp

```
#include "findandreplace.h"

FindAndReplac::FindAndReplac(QWidget *parent) :
    QDialog(parent, Qt::WindowTitleHint | Qt::WindowSystemMenuHint)
{
  FindF = new QLineEdit;
  LFind = new QLabel("Find");
  FindR = new QLineEdit;
  RFind = new QLabel("Replace");
```

```
  btnOk = new QPushButton("OK");
  btnOk->setCheckable(true);

  btnClose = new QPushButton("Close");
  btnClose->setCheckable(true);

  connect(btnOk, SIGNAL(clicked()), SLOT(accept()));
  connect(btnClose, SIGNAL(clicked()), SLOT(reject()));

  QGridLayout* layout = new QGridLayout;
  layout->addWidget(LFind, 0, 0);
  layout->addWidget(FindF, 0, 1);
  layout->addWidget(RFind, 1, 0);
  layout->addWidget(FindR, 1, 1);
  layout->addWidget(btnOk, 2, 0);
  layout->addWidget(btnClose, 2, 1);
  setLayout(layout);

}

QString FindAndReplac::GetFind()
{
 return (FindF->text());
}
QString FindAndReplac::GetReplace()
{
 return (FindR->text());
}
```

A12 texthigligt.h

```
#ifndef TEXTHIGLIGT_H
#define TEXTHIGLIGT_H

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

class QTextDocument;

class Highlighter : public QSyntaxHighlighter
{
     Q_OBJECT

 public:
     Highlighter(QTextDocument *parent = 0);
     void getHigh(QString &standart, int style);

 protected:
     void highlightBlock(const QString &text);

 private:
     struct HighlightingRule
     {
         QRegExp pattern;
         QTextCharFormat format;
```

```
    };
    QVector<HighlightingRule> highlightingRules;

    QRegExp commentStartExpression;
    QRegExp commentEndExpression;

    QTextCharFormat keywordFormat1;
    QTextCharFormat keywordFormat2;
    QTextCharFormat singleLineCommentFormat;
    QTextCharFormat multiLineCommentFormat;
    QTextCharFormat quotationFormat;
    QTextCharFormat tfFormat;

    QTextCharFormat CFormat;
    QTextCharFormat inc;
    QTextCharFormat def;
    QTextCharFormat end;
    QTextCharFormat ifnd;
};

#endif // TEXTHIGLIGT_H
```

A13 texthigligt.cpp

```
#include <QtGui>
#include "texthigligt.h"

 Highlighter::Highlighter(QTextDocument *parent)
     : QSyntaxHighlighter(parent)
 {
    }

void Highlighter::getHigh(QString &standart,int style){
      HighlightingRule rule;
     if (style == 0){
             keywordFormat1.setForeground(Qt::darkGreen);
         keywordFormat1.setFontWeight(QFont::Bold);
     }else if(style == 1) {
         keywordFormat1.setFontItalic(true);
         keywordFormat1.setForeground(Qt::darkGreen);
             keywordFormat1.setFontWeight(QFont::Bold);

     }
     else if (style == 2){
         keywordFormat1.setForeground(Qt::green);
             keywordFormat1.setFontWeight(QFont::Normal);

     }else if (style == 3){
          keywordFormat1.setFontItalic(true);
          keywordFormat1.setForeground(Qt::green);
              keywordFormat1.setFontWeight(QFont::Normal);

     }
     QStringList keywordPatterns1;
     if (standart == "C89" || standart == "C++99/03" || standart ==
"C++11/14"){
     keywordPatterns1 << "\\bchar\\b" << "\\bauto\\b" << "\\bconst\\b"
```

```
                    << "\\bdouble\\b" << "\\benum\\b" <<
"\\bextern\\b"
                    << "\\bfloat\\b" << "\\bint\\b" <<
"\\bregister\\b"
                    << "\\blong\\b" << "\\bshort\\b" <<
"\\bsigned\\b"
            << "\\bstatic\\b" << "\\bstruct\\b" << "\\typedef\\b"
                    << "\\bunion\\b" << "\\bunsigned\\b" <<
"\\bvoid\\b"
                    << "\\bvolatile\\b";
    }
    if (standart == "C++99/03" || standart == "C++11/14"){
        keywordPatterns1 << "\\bbool\\b" << "\\bclass\\b" <<
"\\bexplicit\\b"
                    << "\\bexport\\b" << "\\binline\\b" <<
"\\bmutable\\b"
                    << "\\bnamespace\\b" << "\\btemplate\\b" <<
"\\btypename\\b"
                    << "\\bvirtual\\b" << "\\bwchar_t\\b" <<
"\\bfinal\\b"
            << "\\boverride\\b";
    }
    if (standart == "C++11/14"){
        keywordPatterns1 << "\\balignas\\b" << "\\bauto\\b" <<
"\\bchar16_t\\b"
                     << "\\bchar32_t\\b" << "\\bconstexpr\\b" <<
"\\bdecltype\\b"
                << "\\thread_local\\b";
    }
    foreach (const QString &pattern, keywordPatterns1) {
        rule.pattern = QRegExp(pattern);
        rule.format = keywordFormat1;
        highlightingRules.append(rule);
    }

    if (style == 0){
        keywordFormat2.setForeground(Qt::darkRed);
            keywordFormat2.setFontWeight(QFont::Bold);

    }else if(style == 1) {
        keywordFormat2.setFontItalic(true);

        keywordFormat2.setForeground(Qt::darkRed);
            keywordFormat2.setFontWeight(QFont::Bold);

    }
    else if (style == 2){
        keywordFormat2.setForeground(Qt::red);
            keywordFormat2.setFontWeight(QFont::Normal);

    }else if (style == 3){
        keywordFormat2.setFontItalic(true);
        keywordFormat2.setForeground(Qt::red);
        keywordFormat2.setFontWeight(QFont::Normal);

    }

         QStringList keywordPatterns2;
```

```
    if (standart == "C89" || standart == "C++99/03" || standart ==
"C++11/14"){
    keywordPatterns2 << "\\bbreak\\b" << "\\bcase\\b" <<
"\\bcontinue\\b"
                     << "\\bdefault\\b" << "\\bdo\\b" << "\\else\\b"
                     << "\\bfor\\b" << "\\bgoto\\b" << "\\bif\\b"
                     << "\\breturn\\b" << "\\bsizeof\\b" <<
"\\bswitch\\b"
            << "\\bwhile\\b";
    }
    if (standart == "C++99/03" || standart == "C++11/14"){
        keywordPatterns2 << "\\basm\\b" << "\\bcatch\\b" <<
"\\bconst_cast\\b"
                         << "\\bdelete\\b" << "\\bdynamic_cast\\b" <<
"\\bfriend\\b"
                             << "\\bnew\\b" << "\\boperator\\b" <<
"\\bprivate\\b"
                         << "\\bprotected\\b" << "\\bpublic\\b" <<
"\\breinterpret_cast\\b"
                  << "\\bstatic_cast\\b" << "\\bthis\\b" <<
"\\bthrow\\b"
                  << "\\btry\\b" << "\\btypeid\\b" << "\\busing\\b";

    }
    if (standart == "C++11/14"){
        keywordPatterns2 << "\\balignog\\b" << "\\bnoexcept\\b" <<
"\\bstatic_assert\\b"
            <<"\\band\\b" << "\\band_eq\\b" << "\\bbitand\\b" <<
"\\bbitor\\b"
            <<"\\bcompl\\b" << "\\bnot\\b" << "\\bnot_eq\\b" <<
"\\bor\\b"
            <<"\\bor_eq\\b" << "\\bxor\\b" << "\\bxor_eq\\b";
    }


    foreach (const QString &pattern, keywordPatterns2) {
        rule.pattern = QRegExp(pattern);
        rule.format = keywordFormat2;
        highlightingRules.append(rule);
    }

    if (style == 0){
        def.setForeground(Qt::darkMagenta);
        end.setForeground(Qt::darkMagenta);
        ifnd.setForeground(Qt::darkMagenta);
        inc.setForeground(Qt::darkMagenta);
        CFormat.setForeground(Qt::magenta);
        quotationFormat.setForeground(Qt::magenta);
        tfFormat.setForeground(Qt::magenta);
        singleLineCommentFormat.setForeground(Qt::blue);
            multiLineCommentFormat.setForeground(Qt::blue);
    }else if (style == 1){
        def.setForeground(Qt::darkMagenta);
        end.setForeground(Qt::darkMagenta);
        ifnd.setForeground(Qt::darkMagenta);
        inc.setForeground(Qt::darkMagenta);
        CFormat.setForeground(Qt::magenta);
        quotationFormat.setForeground(Qt::magenta);
```

```
    tfFormat.setForeground(Qt::magenta);
    singleLineCommentFormat.setForeground(Qt::blue);
        multiLineCommentFormat.setForeground(Qt::blue);
    def.setFontWeight(QFont::Bold);
    end.setFontWeight(QFont::Bold);
    ifnd.setFontWeight(QFont::Bold);
    inc.setFontWeight(QFont::Bold);
    CFormat.setFontWeight(QFont::Bold);
    quotationFormat.setFontWeight(QFont::Bold);
    tfFormat.setFontWeight(QFont::Bold);
    singleLineCommentFormat.setFontWeight(QFont::Bold);
    multiLineCommentFormat.setFontWeight(QFont::Bold);
}else if (style == 2){
    def.setForeground(Qt::darkMagenta);
    end.setForeground(Qt::darkMagenta);
    ifnd.setForeground(Qt::darkMagenta);
    inc.setForeground(Qt::darkMagenta);
    CFormat.setForeground(Qt::magenta);
    quotationFormat.setForeground(Qt::magenta);
    tfFormat.setForeground(Qt::magenta);
    singleLineCommentFormat.setForeground(Qt::blue);
        multiLineCommentFormat.setForeground(Qt::blue);
    def.setFontItalic(true);
    end.setFontItalic(true);
    ifnd.setFontItalic(true);
    inc.setFontItalic(true);
    CFormat.setFontItalic(true);
    quotationFormat.setFontItalic(true);
    tfFormat.setFontItalic(true);
    singleLineCommentFormat.setFontItalic(true);
        multiLineCommentFormat.setFontItalic(true);

}else if (style == 3){
    def.setForeground(Qt::darkMagenta);
    end.setForeground(Qt::darkMagenta);
    ifnd.setForeground(Qt::darkMagenta);
    inc.setForeground(Qt::darkMagenta);
    CFormat.setForeground(Qt::magenta);
    quotationFormat.setForeground(Qt::magenta);
    tfFormat.setForeground(Qt::magenta);
    singleLineCommentFormat.setForeground(Qt::blue);
        multiLineCommentFormat.setForeground(Qt::blue);
    def.setFontWeight(QFont::Bold);
    end.setFontWeight(QFont::Bold);
    ifnd.setFontWeight(QFont::Bold);
    inc.setFontWeight(QFont::Bold);
    CFormat.setFontWeight(QFont::Bold);
    quotationFormat.setFontWeight(QFont::Bold);
    tfFormat.setFontWeight(QFont::Bold);
    singleLineCommentFormat.setFontWeight(QFont::Bold);
    multiLineCommentFormat.setFontWeight(QFont::Bold);
    def.setFontItalic(true);
    end.setFontItalic(true);
    ifnd.setFontItalic(true);
    inc.setFontItalic(true);
    CFormat.setFontItalic(true);
    quotationFormat.setFontItalic(true);
    tfFormat.setFontItalic(true);
```

```
        singleLineCommentFormat.setFontItalic(true);
            multiLineCommentFormat.setFontItalic(true);

    }
    rule.pattern = QRegExp("#define [A-Za-z_=()0-9 ]+$");
    rule.format = def;
    highlightingRules.append(rule);

    rule.pattern = QRegExp("#endif");
    rule.format = end;
    highlightingRules.append(rule);

    rule.pattern = QRegExp("#ifndef [A-Za-z_]+$");
    rule.format = ifnd;
    highlightingRules.append(rule);


    rule.pattern = QRegExp("(#include|#pragma once)");
    rule.format = inc;
    highlightingRules.append(rule);

    rule.pattern = QRegExp("[A-Za-z./]+(?=>)");
    rule.format = CFormat;
    highlightingRules.append(rule);

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


    rule.pattern = QRegExp("(\\b[0-9]+\\b|true|false)");
    rule.format = tfFormat;
    highlightingRules.append(rule);


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



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

}


 void Highlighter::highlightBlock(const QString &text)
 {
    foreach (const HighlightingRule &rule, highlightingRules) {
        QRegExp expression(rule.pattern);
        int index = expression.indexIn(text);
        while (index >= 0) {
            int length = expression.matchedLength();
            setFormat(index, length, rule.format);
            index = expression.indexIn(text, index + length);
        }
    }
```

```
        setCurrentBlockState(0);

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

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