НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ імені Ігоря Сікорського»

Кафедра обчислювальної техніки

(повна назва кафедри, циклової комісії)

КУРСОВА РОБОТА

з «Об'єктно орієнтоване програмування»

(назва дисципліни)

на тему: «Текстовий редактор»

Студента 2 курсу ІП-62 групи

напряму підготовки «Програмна інженерія»

спеціальності_____

Павленка В.М.

(прізвище та ініціали)

Керівник_____Порєв В.М.

(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Національна оцінка _____

Кількість балів: _____Оцінка: ECTS _____

Члени комісії

(підпис) (вчене звання, науковий ступінь, прізвище та ініціали)

(підпис) (вчене звання, науковий ступінь, прізвище та ініціали)

Київ - 2017 рік

Національний технічний університет України “КПІ ім. Ігоря Сікорського”

(назва вищого навчального закладу)

Кафедра обчислювальної техніки

Дисципліна Об`єктно орієнтоване програмування

Напрям "Програмна інженерія"

Курс 2 Група ІП-62 Семестр 1

ЗАВДАННЯ

на курсову роботу студента

Павленка Віталія Миколайовича

(прізвище, ім’я, по батькові)

1. Тема роботи Текстовий редактор

2. Строк здачі студентом закінченої роботи

3. Вихідні дані до роботи

4. Зміст розрахунково-пояснювальної записки (перелік питань, які підлягають розробці)

5. Перелік графічного матеріалу (з точним зазначенням обов’язкових креслень)

6. Дата видачі завдання

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Назва етапів курсової роботи	Термін виконання етапів роботи	Підписи керівника, студента
1.	Отримання теми курсової роботи		
2.	Підготовка ТЗ		
3.	Пошук та вивчення літератури з питань курсової роботи		
4.	Розробка алгоритму вирішення задачі		
6.	Узгодження алгоритму з керівником		
5.	Розробка сценарію роботи програми		
6.	Узгодження сценарію роботи програми з керівником		
7.	Узгодження з керівником інтерфейсу користувача		
8.	Розробка програмного забезпечення		
9.	Налагодження розрахункової частини програми		
10.	Розробка та налагодження інтерфейсної частини програми		
11.	Узгодження з керівником набору тестів для контрольного прикладу		
12.	Тестування програми		
13.	Підготовка пояснювальної записки		
14.	Здача курсової роботи на перевірку		
15.	Захист курсової роботи		

Студент _____

(підпис)

Керівник _____ Порєв В.М.

(підпис) (прізвище, ім’я, по батькові)

"___" _____ 2017 р.

АНОТАЦІЯ

Пояснювальна записка до курсової роботи: 57 сторінки, 12 рисунків, 3 таблиці.

Об’єкт дослідження: текстовий редактор.

Мета роботи: реалізування текстового редактора типу Microsoft Word.

Вивчено способи обробки повідомлень та роботу із текстовими даними, приведені змістовні постановки задач, їх індивідуальні моделі, а також описано детальний процес розв’язання кожної з них. Код програми, який дозволяє виконати це, а також детальний опис процесу розв’язання кожної з задач.

Виконана програмна дозволяє редагування тексту, заданого користувачем.

ТЕКСТОВИЙ РЕДАКТОР.

ЗМІСТ

ВСТУП

Сьогодні ми живемо у еру інформаційних технологій, тобто не можливо назвати сферу людської діяльності, де ці технології використовуються. Саме за допомогою комп`ютерів легко виконати розрахунки, їх опрацьовувати, зберігати та обмінюватися ними. Проте кожна сфера потребує особливого програмного забезпечення.

Розробкою таких програм займається спеціальна наука – програмування. З кожним днем ця наука набуває все більшої популярності. Сьогодні майже всі математичні, інженерні, економічні задачі розв`язуються за допомогою спеціальних програм.

Будь-яка робота із комп’ютером включає в себе роботу із текстовими даними. Люди різних галузей створюють на комп’ютерах різні звіти і тому подібні файли та документи. Неможливо представити працівника який би не користування на роботі(не тільки) текстовими редакторами(наприклад Word).

У цій роботі було розроблено програмне забезпечення для роботи із текстовими даними, обробки різних сигналів для роботи із заданими даними.

1 ПОСТАНОВКА ЗАДАЧІ

Розробити програмне забезпечення, для роботи із текстовими даними наприклад Microsoft Word.

Вхідні дані:

Текстові дані, які вводить сам користувач.

Вихідні дані:

Відредагований на бажання самого користувача текс, та за його бажанням збережений у файл(формат теж вибирається за бажанням).

2 ТЕОРЕТИЧНІ ВІДОМОСТІ

З розвитком комп’ютерної техніки змінювався підхід до її програмування. Зараз, коли графічна система присутня майже усюди, а самі програми потребують достатньо складної реалізації, використовують модульний підхід до програмування. Цей підхід передбачає організацію програми, як сукупність невеликих незалежних блоків, що називаються модулями, поведінка та структура яких заздалегідь передбачена. Сам модуль – це функціонально завершений фрагмент програми, який зазвичай оформлюється у вигляді окремого файлу програмного коду.

Діалогова система – це автоматизована система обміну даними між комп’ютером та користувачем, робота якої базується на зверненні по мірі необхідності за інструкціями до користувача, обробку отриманих даних та видачі інформації, що була запитана. Така система також потребувала удосконалення разом з розвитком комп’ютерів. Зараз її функціонал передбачає використання інформації майже будь-якого типу, роботу з нею та вивід результату в різноманітних формах.

З вище описаного зрозуміло, що зараз усі програми використовують разом обидва методи, реалізовуючи модульно-діалогову систему. Яскравим прикладом є форма реєстрації користувача. Ця задача є надзвичайно актуальною у наші часи, адже кожна дія має бути ідентифікована, а для цього потрібна попередня реєстрація. Кожний новий сервіс має свою унікальну структуру цієї реєстраційної форми, що робить кількість варіантів налаштувань однієї задачі дуже великим. Саме цьому основний функціонал такої задачі реалізовується як модульна програма, а специфічні налаштування передаються до неї як вхідні параметри.

Метою даної курсової роботи була саме реалізація текстового редактора з відмінним графічним дизайном.

Поліморфізм означає залежність поведінки від класу, в якому ця поведінка викликається, тобто, два або більше класів можуть реагувати по різному на однакові повідомлення. Це спричинене зміною в одного з класів якогось методу, процедури, функції, шляхом запису іншого алгоритму.

3 АНАЛІЗ МОЖЛИВИХ ВАРІАНТІВ РІШЕННЯ ЗАВДАННЯ.

Оскільки текстовий редактор передбачає використання програми звичайними користувачами практичним варіантом реалізації було б підтримання файлів різних розширень як docx та тому подібні.

Також важливим є не тільки запис текстових даних у різних форматах а і можливість їх зчитування. Підтримка програмою різних форматів дала б поштовх до її розповсюдження та деякої популярності.

На мою думку головною сильною стороною текстового редактору має бути кросплатформеність, тобто підтримання даної програми різними операційними системами як Windows, Linux, MacOS та тому подібне. Що значно розповсюдило б дану програму.

4 ОБГРУНТУВАННЯ ПРОЕКТНОГО РІШЕННЯ

4.1 Загальний алгоритм

1. ПОЧАТОК
2. Відкривається головне вікно програми
3. Введення даних
   - 3.1. Якщо був введений файл то зчитати дані із файлу ІНАКШЕ створити новий тимчасовий файл
4. Зчитати вхідні дані
5. ЯКЩО користувач бажає відредагувати текст
   - 5.1. ЯКЩО був змінений тип шрифту, то змінити на потрібний на виділеному тексті.
   - 5.2. ЯКЩО був змінений колір, то змінити на потрібний колір на виділеному тексті
   - 5.3. ЯКЩО був змінений тип букв, то змінити на потрібний тип на виділеному тексті
   - 5.4. ЯКЩО був змінений тип вид вирівнювання, то змінити на потрібний вид вирівнювання на даній строчці
6. Збереження файлу:
   - 6.1. ЯКЩО користувач бажає сам зберегти файл:
     - 6.1.1. Дати можливість ввести формат вихідного файлу
     - 6.1.2. Дати можливість ввести назву вихідного файлу
   - 6.2. ЯКЩО користувач натиснув на значок закриття програми
     - 6.2.1. Вивести повідомлення про незбереження даного файлу та дати можливість підтвердити свій вчинок або зберегти даний файл
7. КІНЕЦЬ

5 ОПИС ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

5.1 Функціональна структура програмного забезпечення

Функціональну структуру програмного забезпечення представлено на рисунку 5.1.

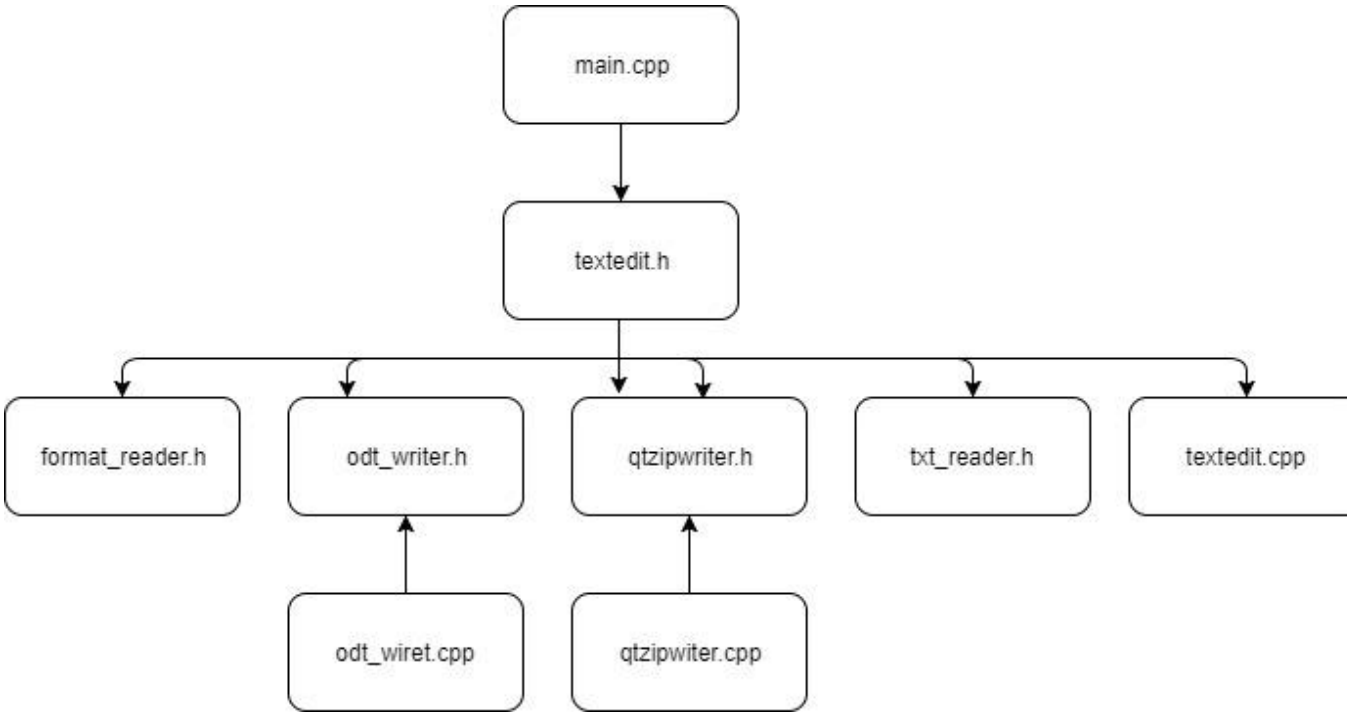

Рисунок 5.1 – Функціональна структура програмного забезпечення

5.2 Опис функцій частин програмного забезпечення

В ході виконання даної роботи було створено наступні модулі та бібліотеки:

а) odt_writer.h – містить клас для запису у форматі odt

б) qtzipwriter.h – містить клас для запису у різних форматах

в) format_writer.h – реалізує збереження різних форматів

г) textedit.h – реалізує графічний інтерфейс та саму логіку текстового редактора

д) txt_reader.h – реалізує зчитування у txt форматі

5.3 Діаграма класів програмного забезпечення (рисунок 5.2.)

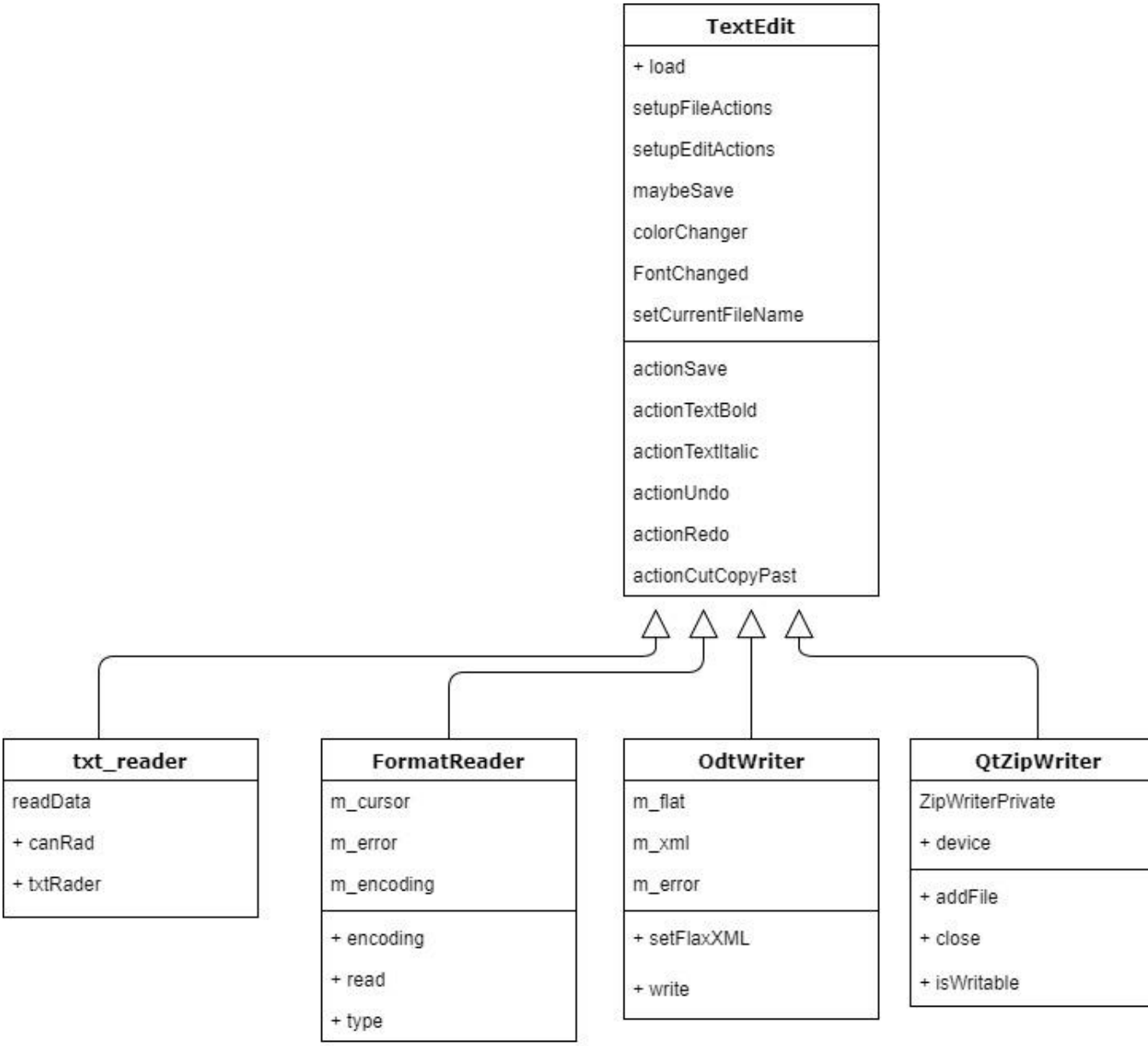

Рисунок 5.2 – Діаграма класів

5.3.1 Реалізація класу

Реалізацію класу TextEdit, описано в таблиці 5.1.

Таблиця 5.1 – реалізація класу TextEdit

№ п/п	Назва функції	Призначення функції	Заголовний файл
1	load	Зчитування файлу	textedit.h
2	fileNew	Створення нового файлу	textedit.h
3	fileOpen	Відкриття файлу	textedit.h
4	fileSave	Збереження файлу	textedit.h
5	fileSave As	Зберегти файл відповідно формату	textedit.h
6	filePrint	Розпечатать файл	textedit.h
7	textBold	Перетворення тексту у жирний вид	textedit.h
8	textUnderline	Підкреслення тексту	textedit.h
9	textItalic	Перетворення тексту на курсив	textedit.h
10	textSize	Змінення розміру текста	textedit.h
11	textColor	Зміна кольору текста	textedit.h
12	textAlign	Зміна розташування текста	textedit.h
13	cursorSelection	Виділення курсора	textedit.h

6 ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

6.1 План тестування

Для запобігання помилок при введенні різних вхідних даних проводиться тестування програми. Важливим є перевірка програми на помилкові вхідні дані, також необхідно впевнитися у вірності роботи редагування текста. Потрібно звернути увагу на такі вхідні дані, що при неправильній обробці можуть призвести до помилок роботи програми чи видати неправильні результати.

Для виявлення усіх можливих виключних ситуацій та перевірки правильності методів програму необхідно запустити на виконання наступних тестів:

а) Тестування правильності роботи з файлами.
   1) Тестування при зчитуванні файлу.
   2) Тестування при збереженні даних у файл.

б) Тестування коректності роботи редагування.
   1) Перевірка коректності роботи зміни кольору.
   2) Перевірка коректності роботи зміни розміру.
   3) Перевірка коректності роботи зміни шрифта.
   4) Перевірка коректності роботи зміни типу букв

6.2 Приклади тестування

6.2.1 Тестування правильності роботи з файлами.

При збереженні даних у файл у різних режимах повинно при перегляді вірно відображати вмістиме файлу. Результат такого тестування наведений на рисунку 6.1.

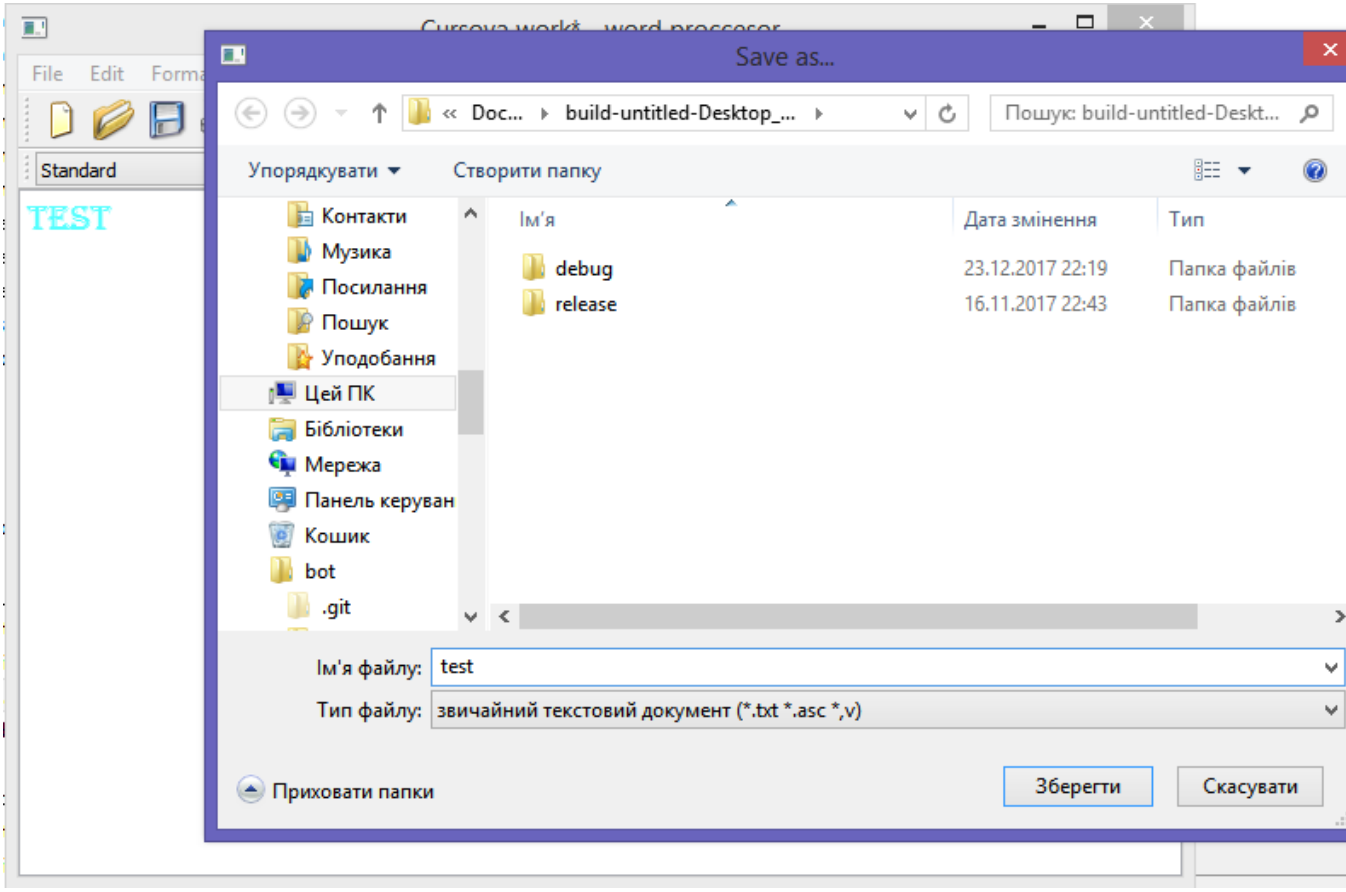

Рисунок 6.1 – Збереження файлу

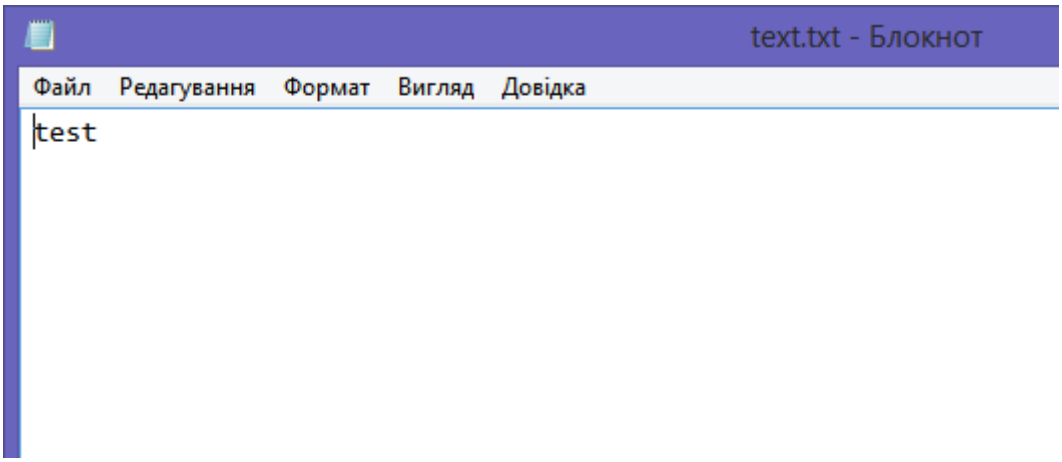

Рисунок 6.2 – Перевірка збереження файлу у форматі .txt

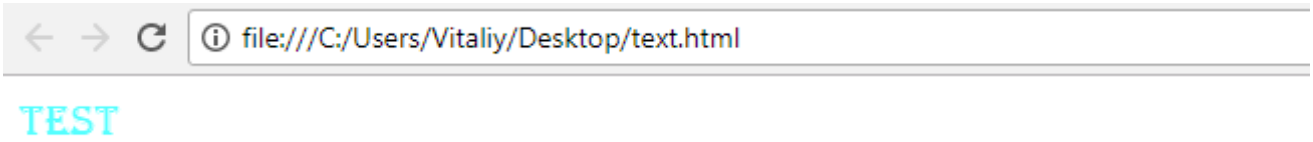

Рисунок 6.3 – Перевірка збереження файлу у форматі .html

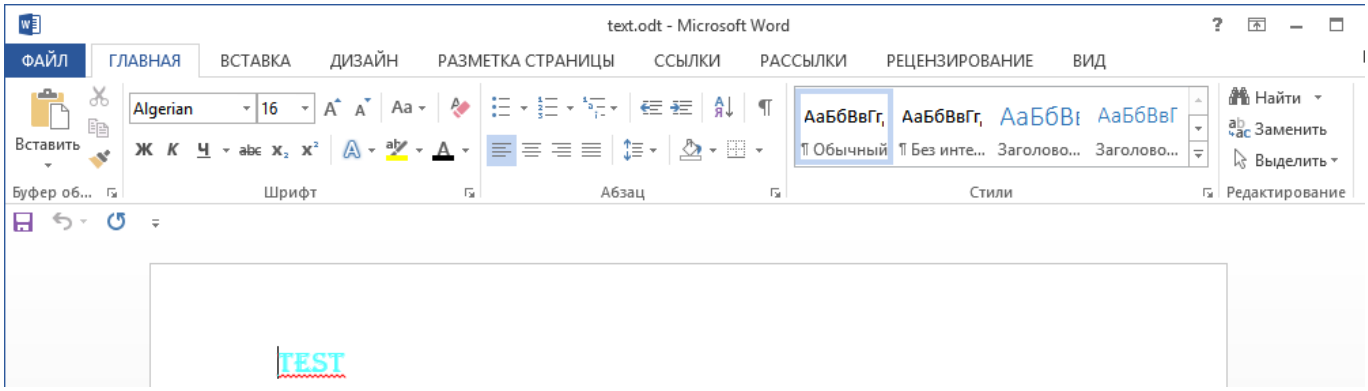

Рисунок 6.3 – Перевірка збереження файлу у форматі .odt

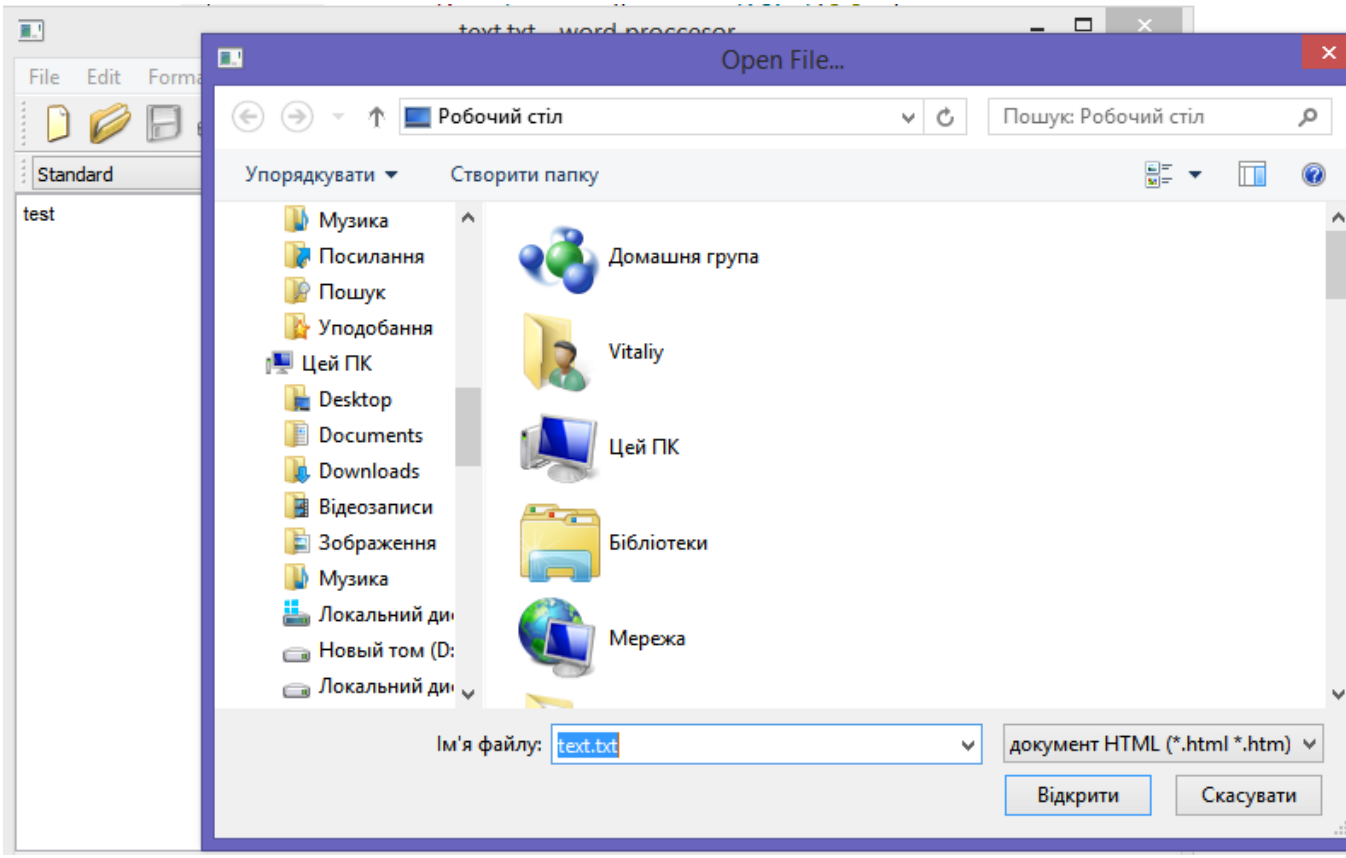

Рисунок 6.4 – Відкриття файлу

6.2.2 Тестування коректності роботи редагування .

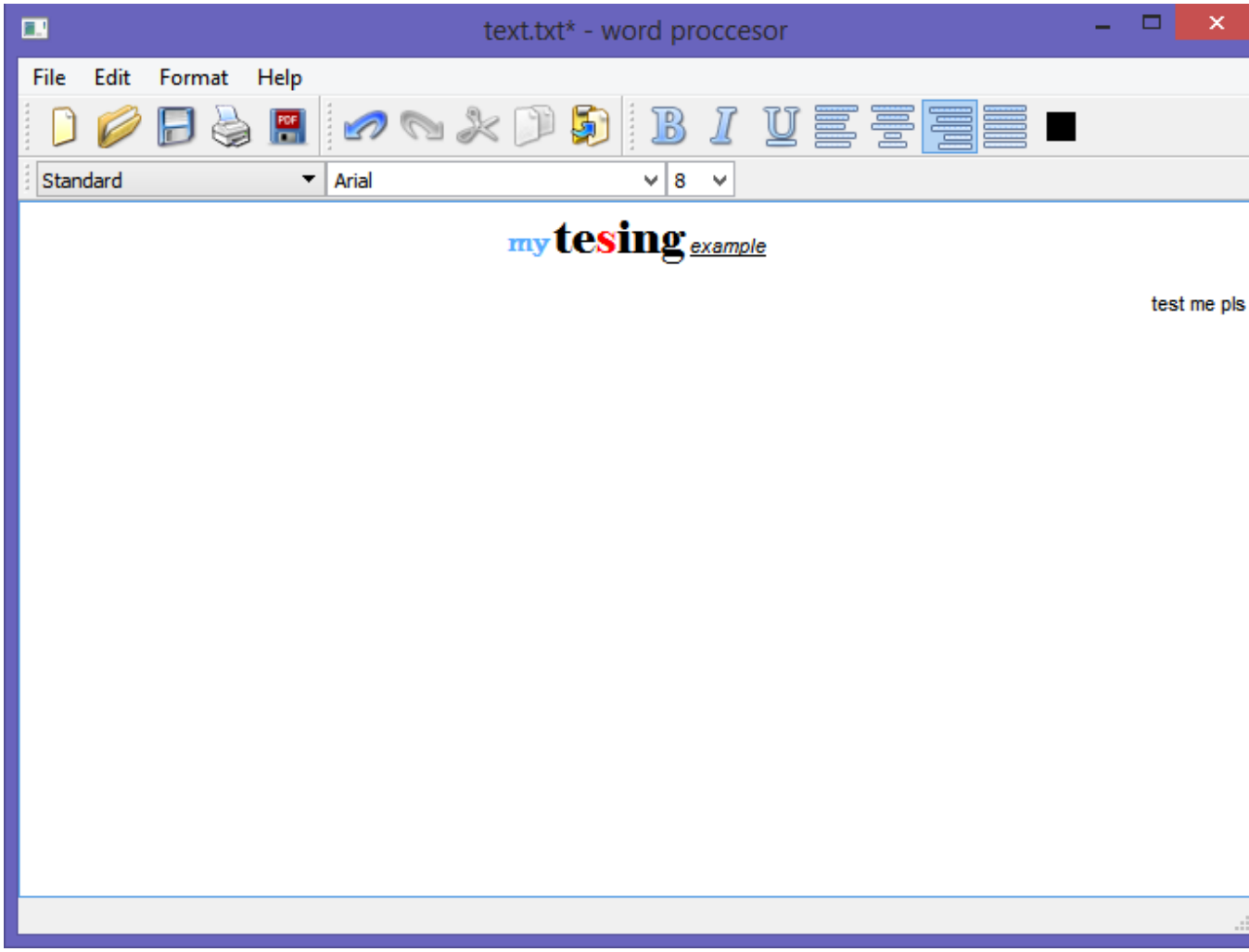

Рисунок 6.5 – Тестування можливостей редагування

7 ІНСТРУКЦІЯ КОРИСТУВАЧА

7.1 Робота з програмою

Після запуску програми (подвійний клік по іконці файлу з розширенням *.exe), відкривається головне вікно програми (Рисунок 7.1).

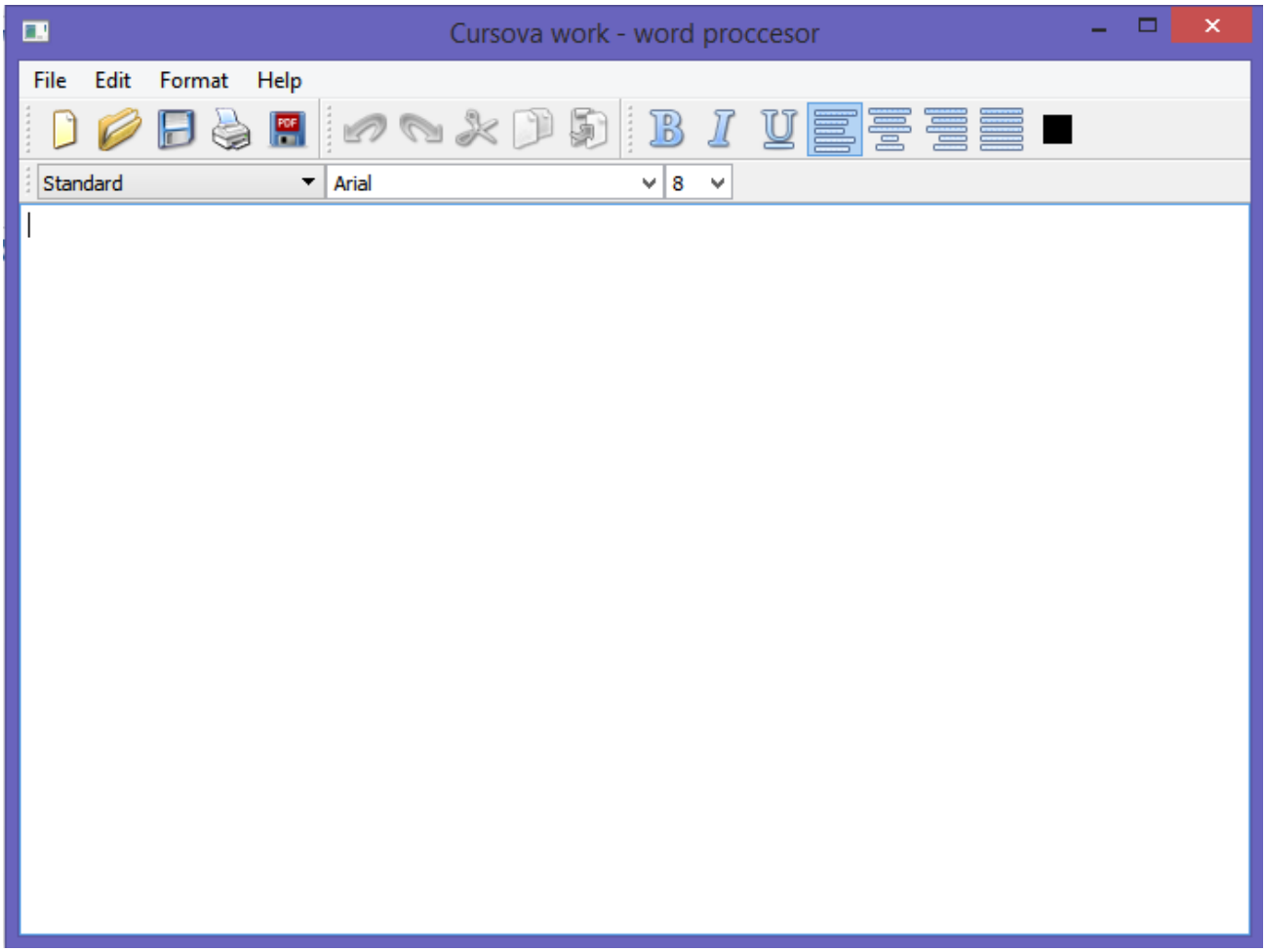

Рисунок 7.1 – Головне вікно програми

Після запуску програми вводьте текст та редагуйте його на ваше бажання . (Рисунок 7.2).

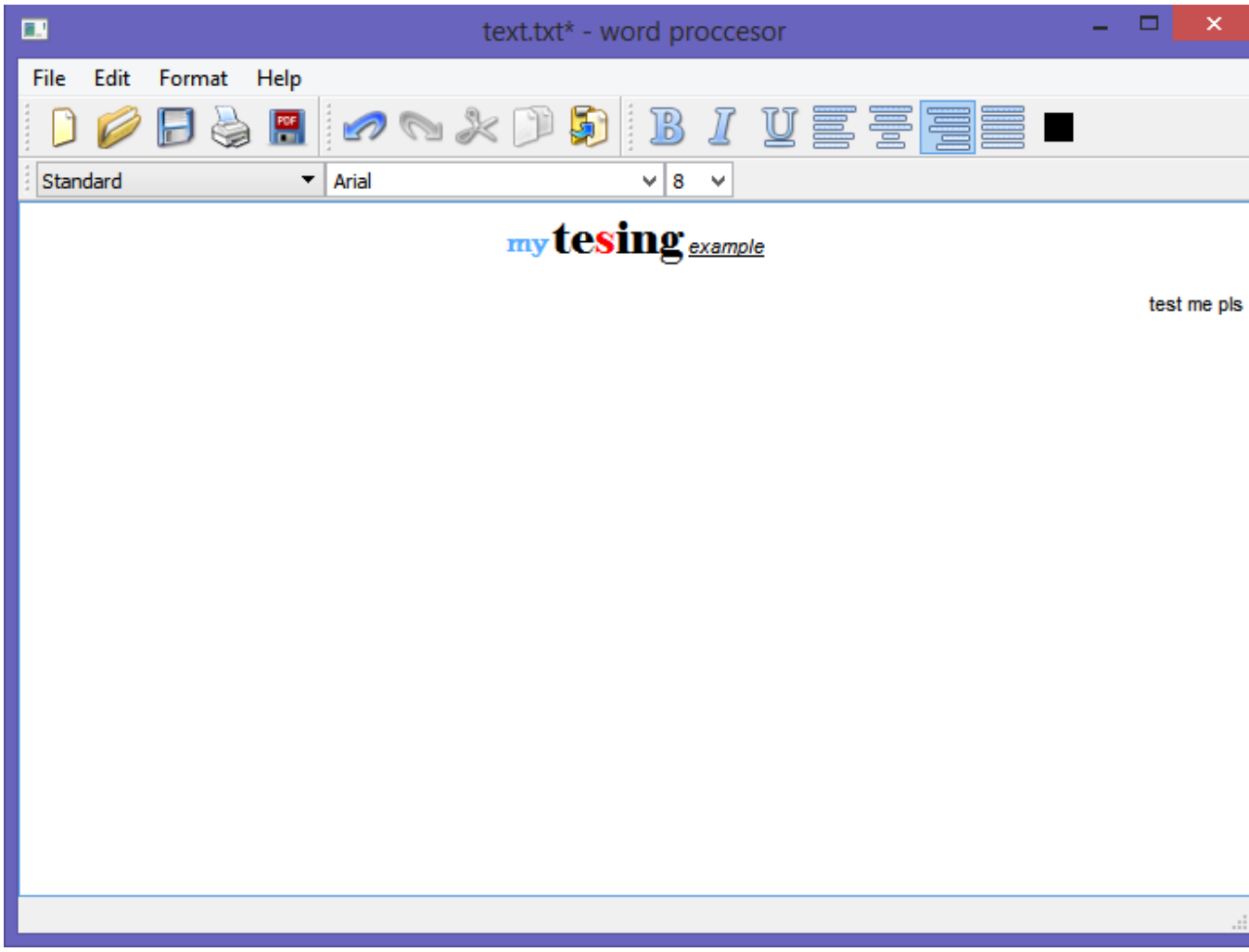

Рисунок 7.2 – Приклад із редагуванням тексту

Наступний крок – зберегти файл для цього потрібно натиснути на File потім на Save As або просто Save який можна викликати із головної програми натиснувни на іконку дискети(Рисунок 7.3 – Збереження змін). У разі коли ви відкрили файл і внесли зміни програма сама Вам допоможе підказавши що ви не зберегли ваш результат, також можна побачити про незбереження даних у назві програми – незбережений файл указується ‘*’ у назві програми.

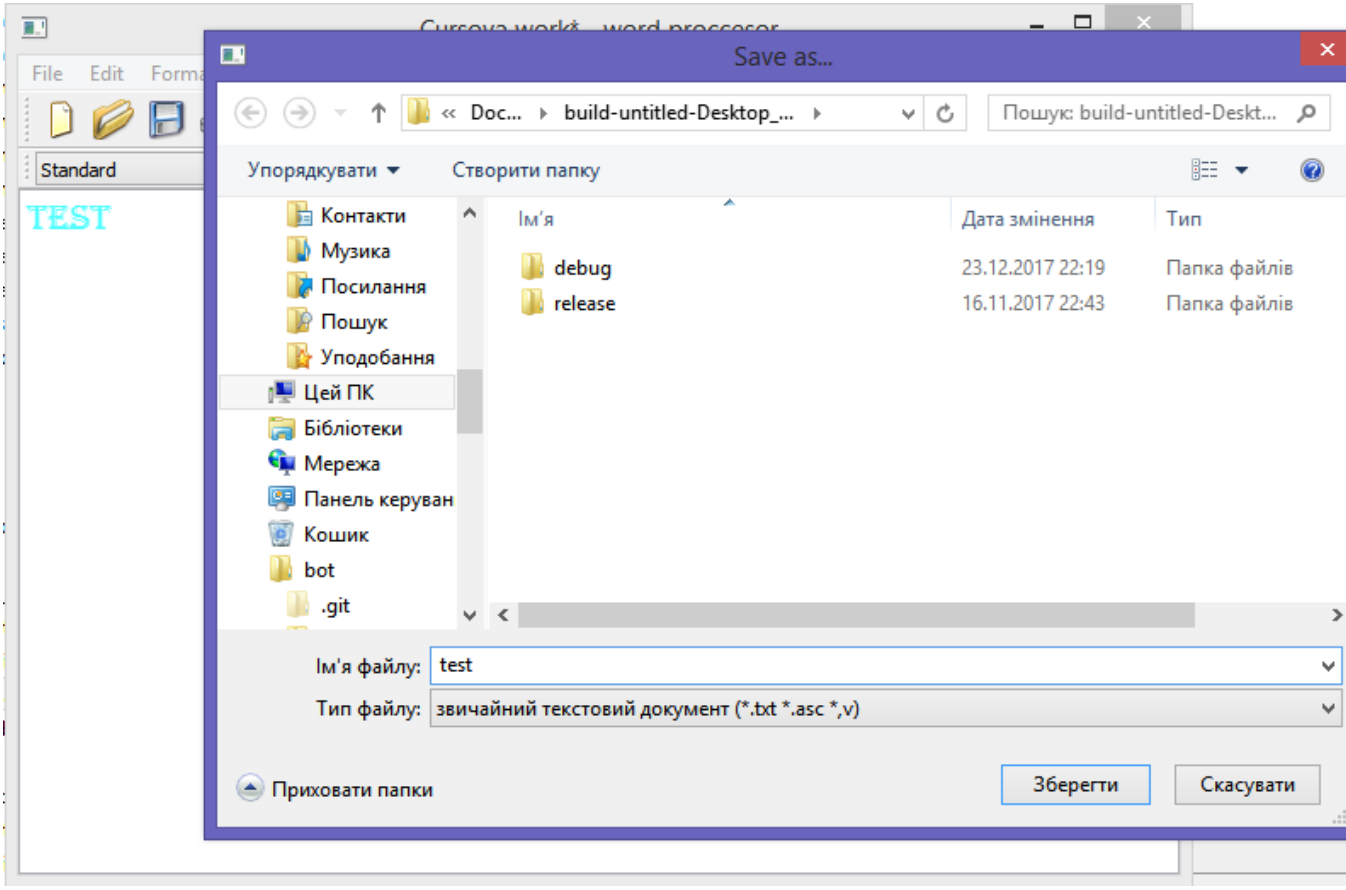

Рисунок 7.3 – Збереження даних

Якщо ви зробили все вірно то файл із заданою вами назвою збережеться у указаному вами місці.

ВИСНОВКИ

Під час виконання даної курсової роботи була розроблена програма – текстовий редактор. На аналітичному етапі було вивчено та досліджено використовувані методи та розроблено загальний алгоритм роботи програми. На прикладі організації цієї програми була опрацьована робота з класами в об’єктно-орієнтовному стилі програмуванні.

Клас — це спеціальна конструкція, яка використовується для групування пов'язаних змінних та функцій. При цьому, глобальні змінні класу члени-змінні називаються полями даних, а також властивостями або атрибутами, а члени-функції називають методами класу. Створений та ініціалізований екземпляр класу називають об'єктом класу. На основі одного класу можна створити безліч об'єктів, що відрізнятимуться один від одного своїм станом значеннями полів.

Важливим є використання інкапсуляції, тобто приховування певних характеристик та методів класів, що забезпечує певну безпеку та можливість безпечної роботи іншими програмістами над даними проектами.

В ході роботи було отримано навички з розробки програмного забезпечення, що базується на графічному відображенні даних. Відповідно до результату та налагодженню роботи програми, результату робота, можна зробити висновок про коректність та достовірність алгоритмів та підходу до вирішення даної задачі.

ТЕКСТИ ПРОГРАМНОГО КОДУ

Файл «main.cpp»

```
#include "textedit.h"

#include <QApplication>
#include <QDesktopWidget>
#include <QCommandLineParser>
#include <QCommandLineOption>

int main(int argc, char *argv[])
{
    Q_INIT_RESOURCE(textedit);

    QApplication a(argc, argv);
    QCoreApplication::setOrganizationName("word proccesor");
    QCoreApplication::setApplicationName("word proccesor");
    QCoreApplication::setApplicationVersion(QT_VERSION_STR);
    QCommandLineParser parser;

parser.setApplicationDescription(QCoreApplication::applicationName());
    parser.addHelpOption();
    parser.addVersionOption();
    parser.addPositionalArgument("file", "The file to open.");
    parser.process(a);

    TextEdit mw;

    const QRect availableGeometry = QApplication::desktop()-
>availableGeometry(&mw);
    mw.resize(availableGeometry.width() / 2,
(availableGeometry.height() * 2) / 3);
    mw.move((availableGeometry.width() - mw.width()) / 2,
            (availableGeometry.height() - mw.height()) / 2);

    if (!mw.load(parser.positionalArguments().value(0,
QLatin1String(":/example.html"))))
        mw.fileNew();

    mw.show();
    return a.exec();
}
```

Файл «textedit.h»

```
#ifndef TEXTEDIT_H
#define TEXTEDIT_H

#include <QMainWindow>
#include <QMap>
#include <QPointer>
```

```
class QAction;
class QComboBox;
class QFontComboBox;
class QTextEdit;
class QTextCharFormat;
class QMenu;
class QPrinter;

class TextEdit : public QMainWindow
{
    Q_OBJECT

public:
    TextEdit(QWidget *parent = 0);

    bool load(const QString &f);

public slots:
    void fileNew();

protected:
    void closeEvent(QCloseEvent *e) override;

private slots:
    void fileOpen();
    bool fileSave();
    bool fileSaveAs();
    void filePrint();
    void filePrintPreview();
    void filePrintPdf();

    void textBold();
    void textUnderline();
    void textItalic();
    void textFamily(const QString &f);
    void textSize(const QString &p);
    void textStyle(int styleIndex);
    void textColor();
    void textAlign(QAction *a);

    void currentCharFormatChanged(const QTextCharFormat &format);
    void cursorPositionChanged();
    void cursorSelection();

    void clipboardDataChanged();
    void about();
    void printPreview(QPrinter *);

private:
    void setupFileActions();
    void setupEditActions();
    void setupTextActions();
    bool maybeSave();
    void setCurrentFileName(const QString &fileName);

    void mergeFormatOnWordOrSelection(const QTextCharFormat &format);
    void fontChanged(const QFont &f);
    void colorChanged(const QColor &c);
    void alignmentChanged(Qt::Alignment a);

    QAction *actionSave;
    QAction *actionTextBold;
```

```
    QAction *actionTextUnderline;
    QAction *actionTextItalic;
    QAction *actionTextColor;
    QAction *actionAlignLeft;
    QAction *actionAlignCenter;
    QAction *actionAlignRight;
    QAction *actionAlignJustify;
    QAction *actionUndo;
    QAction *actionRedo;
//#ifndef QT_NO_CLIPBOARD
    QAction *actionCut;
    QAction *actionCopy;
    QAction *actionPaste;
//#endif

    QComboBox *comboStyle;
    QFontComboBox *comboFont;
    QComboBox *comboSize;

    QToolBar *tb;
    QString fileName;
    QTextEdit *textEdit;
};

#endif // TEXTEDIT_H
```

Файл «textedit.cpp»

```
#include <QAction>
#include <QApplication>
#include <QClipboard>
#include <QColorDialog>
#include <QComboBox>
#include <QFontComboBox>
#include <QFile>
#include <QFileDialog>
#include <QFileInfo>
#include <QFontDatabase>
#include <QMenu>
#include <QMenuBar>
#include <QTextCodec>
#include <QTextEdit>
#include <QStatusBar>
#include <QToolBar>
#include <QTextCursor>
#include <QTextDocumentWriter>
#include <QTextList>
#include <QtDebug>
#include <QCloseEvent>
#include <QMessageBox>
#include <QMimeData>
#if defined(QT_PRINTSUPPORT_LIB)
#include <QtPrintSupport/qtprintsupportglobal.h>
#if QT_CONFIG(printer)
#if QT_CONFIG(printdialog)
#include <QPrintDialog>
#endif
#include <QPrinter>
#if QT_CONFIG(printpreviewdialog)
#include <QPrintPreviewDialog>
#endif
#endif
#endif
```

```
#include "textedit.h"

const QString rsrcPath = ":/";


TextEdit::TextEdit(QWidget *parent)
    : QMainWindow(parent)
{
    textEdit = new QTextEdit(this);
    connect(textEdit, &QTextEdit::currentCharFormatChanged,
            this, &TextEdit::currentCharFormatChanged);
    connect(textEdit, &QTextEdit::cursorPositionChanged,
            this, &TextEdit::cursorPositionChanged);
    connect(textEdit, &QTextEdit::selectionChanged,
            this, &TextEdit::cursorSelection);

    setCentralWidget(textEdit);

    setToolButtonStyle(Qt::ToolButtonFollowStyle);
    setupFileActions();
    setupEditActions();
    setupTextActions();

    {
        QMenu *helpMenu = menuBar()->addMenu(tr("Help"));
        helpMenu->addAction(tr("About"), this, &TextEdit::about);
    }

    QFont textFont("Helvetica");
    textFont.setStyleHint(QFont::SansSerif);
    textEdit->setFont(textFont);
    fontChanged(textEdit->font());
    colorChanged(textEdit->textColor());
    alignmentChanged(textEdit->alignment());

    connect(textEdit->document(), &QTextDocument::modificationChanged,
            actionSave, &QAction::setEnabled);
    connect(textEdit->document(), &QTextDocument::modificationChanged,
            this, &QWidget::setWindowModified);
    connect(textEdit->document(), &QTextDocument::undoAvailable,
            actionUndo, &QAction::setEnabled);
    connect(textEdit->document(), &QTextDocument::redoAvailable,
            actionRedo, &QAction::setEnabled);

    setWindowModified(textEdit->document()->isModified());
    actionSave->setEnabled(textEdit->document()->isModified());
    actionUndo->setEnabled(textEdit->document()->isUndoAvailable());
    actionRedo->setEnabled(textEdit->document()->isRedoAvailable());

    actionCut->setEnabled(false);
    actionCopy->setEnabled(false);

    connect(QApplication::clipboard(), &QClipboard::dataChanged, this,
&TextEdit::clipboardDataChanged);


    textEdit->setFocus();
    setCurrentFileName(QString());
}

void TextEdit::closeEvent(QCloseEvent *e)
{
    if (maybeSave())
```

```
        e->accept();
    else
        e->ignore();
}

void TextEdit::setupFileActions()
{
    QToolBar *tb = addToolBar(tr("File Actions"));
    QMenu *menu = menuBar()->addMenu(tr("&File"));

    const QIcon newIcon = QIcon::fromTheme("document-new", QIcon(rsrcPath +
"/filenew.png"));
    QAction *a = menu->addAction(newIcon,  tr("&New"), this, &TextEdit::fileNew);
    tb->addAction(a);
    a->setPriority(QAction::LowPriority);
    a->setShortcut(QKeySequence::New);

    const QIcon openIcon = QIcon::fromTheme("document-open", QIcon(rsrcPath +
"/fileopen.png"));
    a = menu->addAction(openIcon, tr("&Open..."), this, &TextEdit::fileOpen);
    a->setShortcut(QKeySequence::Open);
    tb->addAction(a);

    menu->addSeparator();

    const QIcon saveIcon = QIcon::fromTheme("document-save", QIcon(rsrcPath +
"/filesave.png"));
    actionSave = menu->addAction(saveIcon, tr("&Save"), this, &TextEdit::fileSave);
    actionSave->setShortcut(QKeySequence::Save);
    actionSave->setEnabled(false);
    tb->addAction(actionSave);

    a = menu->addAction(tr("Save &As..."), this, &TextEdit::fileSaveAs);
    a->setPriority(QAction::LowPriority);
    menu->addSeparator();

#ifndef QT_NO_PRINTER
    const QIcon printIcon = QIcon::fromTheme("document-print", QIcon(rsrcPath +
"/fileprint.png"));
    a = menu->addAction(printIcon, tr("&Print..."), this, &TextEdit::filePrint);
    a->setPriority(QAction::LowPriority);
    a->setShortcut(QKeySequence::Print);
    tb->addAction(a);

    const QIcon filePrintIcon = QIcon::fromTheme("fileprint", QIcon(rsrcPath +
"/fileprint.png"));
    menu->addAction(filePrintIcon, tr("Print Preview..."), this,
&TextEdit::filePrintPreview);

    const QIcon exportPdfIcon = QIcon::fromTheme("exportpdf", QIcon(rsrcPath +
"/exportpdf.png"));
    a = menu->addAction(exportPdfIcon, tr("&Export PDF..."), this,
&TextEdit::filePrintPdf);
    a->setPriority(QAction::LowPriority);
    a->setShortcut(Qt::CTRL + Qt::Key_D);
    tb->addAction(a);

    menu->addSeparator();
#endif

    a = menu->addAction(tr("&Quit"), this, &QWidget::close);
    a->setShortcut(Qt::CTRL + Qt::Key_Q);
}
```

```
void TextEdit::setupEditActions()
{
    QToolBar *tb = addToolBar(tr("Edit Actions"));
    QMenu *menu = menuBar()->addMenu(tr("&Edit"));

    const QIcon undoIcon = QIcon::fromTheme("edit-undo", QIcon(rsrcPath +
"/editundo.png"));
    actionUndo = menu->addAction(undoIcon, tr("&Undo"), textEdit, &QTextEdit::undo);
    actionUndo->setShortcut(QKeySequence::Undo);
    tb->addAction(actionUndo);

    const QIcon redoIcon = QIcon::fromTheme("edit-redo", QIcon(rsrcPath +
"/editredo.png"));
    actionRedo = menu->addAction(redoIcon, tr("&Redo"), textEdit, &QTextEdit::redo);
    actionRedo->setPriority(QAction::LowPriority);
    actionRedo->setShortcut(QKeySequence::Redo);
    tb->addAction(actionRedo);
    menu->addSeparator();

#ifndef QT_NO_CLIPBOARD
    const QIcon cutIcon = QIcon::fromTheme("edit-cut", QIcon(rsrcPath +
"/editcut.png"));
    actionCut = menu->addAction(cutIcon, tr("Cu&t"), textEdit, &QTextEdit::cut);
    actionCut->setPriority(QAction::LowPriority);
    actionCut->setShortcut(QKeySequence::Cut);
    tb->addAction(actionCut);

    const QIcon copyIcon = QIcon::fromTheme("edit-copy", QIcon(rsrcPath +
"/editcopy.png"));
    actionCopy = menu->addAction(copyIcon, tr("&Copy"), textEdit, &QTextEdit::copy);
    actionCopy->setPriority(QAction::LowPriority);
    actionCopy->setShortcut(QKeySequence::Copy);
    tb->addAction(actionCopy);

    const QIcon pasteIcon = QIcon::fromTheme("edit-paste", QIcon(rsrcPath +
"/editpaste.png"));
    actionPaste = menu->addAction(pasteIcon, tr("&Paste"), textEdit,
&QTextEdit::paste);
    actionPaste->setPriority(QAction::LowPriority);
    actionPaste->setShortcut(QKeySequence::Paste);
    tb->addAction(actionPaste);
    if (const QMimeData *md = QApplication::clipboard()->mimeData())
        actionPaste->setEnabled(md->hasText());
#endif
}

void TextEdit::setupTextActions()
{
    QToolBar *tb = addToolBar(tr("Format Actions"));
    QMenu *menu = menuBar()->addMenu(tr("F&ormat"));

    const QIcon boldIcon = QIcon::fromTheme("format-text-bold", QIcon(rsrcPath +
"/textbold.png"));
    actionTextBold = menu->addAction(boldIcon, tr("&Bold"), this,
&TextEdit::textBold);
    actionTextBold->setShortcut(Qt::CTRL + Qt::Key_B);
    actionTextBold->setPriority(QAction::LowPriority);
    QFont bold;
    bold.setBold(true);
    actionTextBold->setFont(bold);
    tb->addAction(actionTextBold);
    actionTextBold->setCheckable(true);
```

```
    const QIcon italicIcon = QIcon::fromTheme("format-text-italic", QIcon(rsrcPath +
"/textitalic.png"));
    actionTextItalic = menu->addAction(italicIcon, tr("&Italic"), this,
&TextEdit::textItalic);
    actionTextItalic->setPriority(QAction::LowPriority);
    actionTextItalic->setShortcut(Qt::CTRL + Qt::Key_I);
    QFont italic;
    italic.setItalic(true);
    actionTextItalic->setFont(italic);
    tb->addAction(actionTextItalic);
    actionTextItalic->setCheckable(true);

    const QIcon underlineIcon = QIcon::fromTheme("format-text-underline",
QIcon(rsrcPath + "/textunder.png"));
    actionTextUnderline = menu->addAction(underlineIcon, tr("&Underline"), this,
&TextEdit::textUnderline);
    actionTextUnderline->setShortcut(Qt::CTRL + Qt::Key_U);
    actionTextUnderline->setPriority(QAction::LowPriority);
    QFont underline;
    underline.setUnderline(true);
    actionTextUnderline->setFont(underline);
    tb->addAction(actionTextUnderline);
    actionTextUnderline->setCheckable(true);

    menu->addSeparator();

    const QIcon leftIcon = QIcon::fromTheme("format-justify-left", QIcon(rsrcPath +
"/textleft.png"));
    actionAlignLeft = new QAction(leftIcon, tr("&Left"), this);
    actionAlignLeft->setShortcut(Qt::CTRL + Qt::Key_L);
    actionAlignLeft->setCheckable(true);
    actionAlignLeft->setPriority(QAction::LowPriority);
    const QIcon centerIcon = QIcon::fromTheme("format-justify-center", QIcon(rsrcPath
+ "/textcenter.png"));
    actionAlignCenter = new QAction(centerIcon, tr("C&enter"), this);
    actionAlignCenter->setShortcut(Qt::CTRL + Qt::Key_E);
    actionAlignCenter->setCheckable(true);
    actionAlignCenter->setPriority(QAction::LowPriority);
    const QIcon rightIcon = QIcon::fromTheme("format-justify-right", QIcon(rsrcPath +
"/textright.png"));
    actionAlignRight = new QAction(rightIcon, tr("&Right"), this);
    actionAlignRight->setShortcut(Qt::CTRL + Qt::Key_R);
    actionAlignRight->setCheckable(true);
    actionAlignRight->setPriority(QAction::LowPriority);
    const QIcon fillIcon = QIcon::fromTheme("format-justify-fill", QIcon(rsrcPath +
"/textjustify.png"));
    actionAlignJustify = new QAction(fillIcon, tr("&Justify"), this);
    actionAlignJustify->setShortcut(Qt::CTRL + Qt::Key_J);
    actionAlignJustify->setCheckable(true);
    actionAlignJustify->setPriority(QAction::LowPriority);

    // Make sure the alignLeft  is always left of the alignRight
    QActionGroup *alignGroup = new QActionGroup(this);
    connect(alignGroup, &QActionGroup::triggered, this, &TextEdit::textAlign);

    if (QApplication::isLeftToRight()) {
        alignGroup->addAction(actionAlignLeft);
        alignGroup->addAction(actionAlignCenter);
        alignGroup->addAction(actionAlignRight);
    } else {
        alignGroup->addAction(actionAlignRight);
        alignGroup->addAction(actionAlignCenter);
```

```
        alignGroup->addAction(actionAlignLeft);
    }
    alignGroup->addAction(actionAlignJustify);

    tb->addActions(alignGroup->actions());
    menu->addActions(alignGroup->actions());

    menu->addSeparator();

    QPixmap pix(16, 16);
    pix.fill(Qt::black);
    actionTextColor = menu->addAction(pix, tr("&Color..."), this,
&TextEdit::textColor);
    tb->addAction(actionTextColor);

    tb = addToolBar(tr("Format Actions"));
    tb->setAllowedAreas(Qt::TopToolBarArea | Qt::BottomToolBarArea);
    addToolBarBreak(Qt::TopToolBarArea);
    addToolBar(tb);

    comboStyle = new QComboBox(tb);
    tb->addWidget(comboStyle);
    comboStyle->addItem("Standard");
    comboStyle->addItem("Bullet List (Disc)");
    comboStyle->addItem("Bullet List (Circle)");
    comboStyle->addItem("Bullet List (Square)");
    comboStyle->addItem("Ordered List (Decimal)");
    comboStyle->addItem("Ordered List (Alpha lower)");
    comboStyle->addItem("Ordered List (Alpha upper)");
    comboStyle->addItem("Ordered List (Roman lower)");
    comboStyle->addItem("Ordered List (Roman upper)");

    connect(comboStyle, QOverload<int>::of(&QComboBox::activated), this,
&TextEdit::textStyle);

    comboFont = new QFontComboBox(tb);
    tb->addWidget(comboFont);
    connect(comboFont, QOverload<const QString &>::of(&QComboBox::activated), this,
&TextEdit::textFamily);

    comboSize = new QComboBox(tb);
    comboSize->setObjectName("comboSize");
    tb->addWidget(comboSize);
    comboSize->setEditable(true);

    const QList<int> standardSizes = QFontDatabase::standardSizes();
    foreach (int size, standardSizes)
        comboSize->addItem(QString::number(size));
    comboSize-
>setCurrentIndex(standardSizes.indexOf(QApplication::font().pointSize()));

    connect(comboSize, QOverload<const QString &>::of(&QComboBox::activated), this,
&TextEdit::textSize);
}

bool TextEdit::load(const QString &f)
{
    if (!QFile::exists(f))
        return false;
    QFile file(f);
    if (!file.open(QFile::ReadOnly))
        return false;
```

```
    QByteArray data = file.readAll();
    QTextCodec *codec = Qt::codecForHtml(data);
    QString str = codec->toUnicode(data);
    if (Qt::mightBeRichText(str)) {
        textEdit->setHtml(str);
    } else {
        str = QString::fromLocal8Bit(data);
        textEdit->setPlainText(str);
    }

    setCurrentFileName(f);
    return true;
}

bool TextEdit::maybeSave()
{
    if (!textEdit->document()->isModified())
        return true;

    const QMessageBox::StandardButton ret =
        QMessageBox::warning(this, tr("Cursova Work"),
                             tr("The document has been modified.\n"
                                "Do you want to save your changes?"),
                             QMessageBox::Save | QMessageBox::Discard |
QMessageBox::Cancel);
    if (ret == QMessageBox::Save)
        return fileSave();
    else if (ret == QMessageBox::Cancel)
        return false;
    return true;
}

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

    QString shownName;
    if (fileName.isEmpty())
        shownName = "Cursova work";
    else
        shownName = QFileInfo(fileName).fileName();

    setWindowTitle(tr("%1[*] - %2").arg(shownName,
QCoreApplication::applicationName()));
    setWindowModified(false);
}

void TextEdit::fileNew()
{
    if (maybeSave()) {
        textEdit->clear();
        setCurrentFileName(QString());
    }
}

void TextEdit::fileOpen()
{
    QFileDialog fileDialog(this, tr("Open File..."));
    fileDialog.setAcceptMode(QFileDialog::AcceptOpen);
    fileDialog.setFileMode(QFileDialog::ExistingFile);
    fileDialog.setMimeTypeFilters(QStringList() << "text/html" << "text/plain");
    if (fileDialog.exec() != QDialog::Accepted)
```

```
        return;
    const QString fn = fileDialog.selectedFiles().first();
    if (load(fn))
        statusBar()->showMessage(tr("Opened
\"%1\"").arg(QDir::toNativeSeparators(fn)));
    else
        statusBar()->showMessage(tr("Could not open
\"%1\"").arg(QDir::toNativeSeparators(fn)));
}

bool TextEdit::fileSave()
{
    if (fileName.isEmpty())
        return fileSaveAs();
    if (fileName.startsWith(QStringLiteral(":/")))
        return fileSaveAs();

    QTextDocumentWriter writer(fileName);
    bool success = writer.write(textEdit->document());
    if (success) {
        textEdit->document()->setModified(false);
        statusBar()->showMessage(tr("Wrote
\"%1\"").arg(QDir::toNativeSeparators(fileName)));
    } else {
        statusBar()->showMessage(tr("Could not write to file \"%1\"")
                                 .arg(QDir::toNativeSeparators(fileName)));
    }
    return success;
}

bool TextEdit::fileSaveAs()
{
    QFileDialog fileDialog(this, tr("Save as..."));
    fileDialog.setAcceptMode(QFileDialog::AcceptSave);
    QStringList mimeTypes;
    mimeTypes << "application/vnd.oasis.opendocument.text" << "text/html" <<
"text/plain";
    fileDialog.setMimeTypeFilters(mimeTypes);
    fileDialog.setDefaultSuffix("odt");
    if (fileDialog.exec() != QDialog::Accepted)
        return false;
    const QString fn = fileDialog.selectedFiles().first();
    setCurrentFileName(fn);
    return fileSave();
}

void TextEdit::filePrint()
{
#if QT_CONFIG(printdialog)
    QPrinter printer(QPrinter::HighResolution);
    QPrintDialog *dlg = new QPrintDialog(&printer, this);
    if (textEdit->textCursor().hasSelection())
        dlg->addEnabledOption(QAbstractPrintDialog::PrintSelection);
    dlg->setWindowTitle(tr("Print Document"));
    if (dlg->exec() == QDialog::Accepted)
        textEdit->print(&printer);
    delete dlg;
#endif
}

void TextEdit::filePrintPreview()
{
#if QT_CONFIG(printpreviewdialog)
```

```
    QPrinter printer(QPrinter::HighResolution);
    QPrintPreviewDialog preview(&printer, this);
    connect(&preview, &QPrintPreviewDialog::paintRequested, this,
&TextEdit::printPreview);
    preview.exec();
#endif
}

void TextEdit::printPreview(QPrinter *printer)
{
#ifdef QT_NO_PRINTER
    Q_UNUSED(printer);
#else
    textEdit->print(printer);
#endif
}

void TextEdit::filePrintPdf()
{
#ifndef QT_NO_PRINTER
    QFileDialog fileDialog(this, tr("Export PDF"));
    fileDialog.setAcceptMode(QFileDialog::AcceptSave);
    fileDialog.setMimeTypeFilters(QStringList("application/pdf"));
    fileDialog.setDefaultSuffix("pdf");
    if (fileDialog.exec() != QDialog::Accepted)
        return;
    QString fileName = fileDialog.selectedFiles().first();
    QPrinter printer(QPrinter::HighResolution);
    printer.setOutputFormat(QPrinter::PdfFormat);
    printer.setOutputFileName(fileName);
    textEdit->document()->print(&printer);
    statusBar()->showMessage(tr("Exported \"%1\"")
                             .arg(QDir::toNativeSeparators(fileName)));
#endif
}

void TextEdit::textBold()
{
    QTextCharFormat fmt;
    fmt.setFontWeight(actionTextBold->isChecked() ? QFont::Bold : QFont::Normal);
    mergeFormatOnWordOrSelection(fmt);
}

void TextEdit::textUnderline()
{
    QTextCharFormat fmt;
    fmt.setFontUnderline(actionTextUnderline->isChecked());
    mergeFormatOnWordOrSelection(fmt);
}

void TextEdit::textItalic()
{
    QTextCharFormat fmt;
    fmt.setFontItalic(actionTextItalic->isChecked());
    mergeFormatOnWordOrSelection(fmt);
}

void TextEdit::textFamily(const QString &f)
{
    QTextCharFormat fmt;
    fmt.setFontFamily(f);
    mergeFormatOnWordOrSelection(fmt);
}
```

```
void TextEdit::textSize(const QString &p)
{
    qreal pointSize = p.toFloat();
    if (p.toFloat() > 0) {
        QTextCharFormat fmt;
        fmt.setFontPointSize(pointSize);
        mergeFormatOnWordOrSelection(fmt);
    }
}

void TextEdit::textStyle(int styleIndex)
{
    QTextCursor cursor = textEdit->textCursor();

    if (styleIndex != 0) {
        QTextListFormat::Style style = QTextListFormat::ListDisc;

        switch (styleIndex) {
            default:
            case 1:
                style = QTextListFormat::ListDisc;
                break;
            case 2:
                style = QTextListFormat::ListCircle;
                break;
            case 3:
                style = QTextListFormat::ListSquare;
                break;
            case 4:
                style = QTextListFormat::ListDecimal;
                break;
            case 5:
                style = QTextListFormat::ListLowerAlpha;
                break;
            case 6:
                style = QTextListFormat::ListUpperAlpha;
                break;
            case 7:
                style = QTextListFormat::ListLowerRoman;
                break;
            case 8:
                style = QTextListFormat::ListUpperRoman;
                break;
        }

        cursor.beginEditBlock();

        QTextBlockFormat blockFmt = cursor.blockFormat();

        QTextListFormat listFmt;

        if (cursor.currentList()) {
            listFmt = cursor.currentList()->format();
        } else {
            listFmt.setIndent(blockFmt.indent() + 1);
            blockFmt.setIndent(0);
            cursor.setBlockFormat(blockFmt);
        }

        listFmt.setStyle(style);

        cursor.createList(listFmt);
```

```
        cursor.endEditBlock();
    } else {
        QTextBlockFormat bfmt;
        bfmt.setObjectIndex(-1);
        cursor.mergeBlockFormat(bfmt);
    }
}

void TextEdit::textColor()
{
    QColor col = QColorDialog::getColor(textEdit->textColor(), this);
    if (!col.isValid())
        return;
    QTextCharFormat fmt;
    fmt.setForeground(col);
    mergeFormatOnWordOrSelection(fmt);
    colorChanged(col);
}

void TextEdit::textAlign(QAction *a)
{
    if (a == actionAlignLeft)
        textEdit->setAlignment(Qt::AlignLeft | Qt::AlignAbsolute);
    else if (a == actionAlignCenter)
        textEdit->setAlignment(Qt::AlignHCenter);
    else if (a == actionAlignRight)
        textEdit->setAlignment(Qt::AlignRight | Qt::AlignAbsolute);
    else if (a == actionAlignJustify)
        textEdit->setAlignment(Qt::AlignJustify);
}

void TextEdit::currentCharFormatChanged(const QTextCharFormat &format)
{
    fontChanged(format.font());
    colorChanged(format.foreground().color());
}

void TextEdit::cursorPositionChanged()
{
    alignmentChanged(textEdit->alignment());
}

void TextEdit::cursorSelection()
{
    QTextCursor cursor = textEdit->textCursor();
    if (!cursor.hasSelection()) {
        actionCut->setEnabled(false);
        actionCopy->setEnabled(false);
    } else {
        actionCut->setEnabled(true);
        actionCopy->setEnabled(true);
    }
}

void TextEdit::clipboardDataChanged()
{
#ifndef QT_NO_CLIPBOARD
    if (const QMimeData *md = QApplication::clipboard()->mimeData())
        actionPaste->setEnabled(md->hasText());
#endif
}
```

```
void TextEdit::about()
{
    QMessageBox::about(this, tr("About"), tr("This is cursova work "
        "made by Vitaliy Pavlenko using Qt."));
}

void TextEdit::mergeFormatOnWordOrSelection(const QTextCharFormat &format)
{
    QTextCursor cursor = textEdit->textCursor();
    if (!cursor.hasSelection())
        cursor.select(QTextCursor::WordUnderCursor);
    cursor.mergeCharFormat(format);
    textEdit->mergeCurrentCharFormat(format);
}

void TextEdit::fontChanged(const QFont &f)
{
    comboFont->setCurrentIndex(comboFont->findText(QFontInfo(f).family()));
    comboSize->setCurrentIndex(comboSize->findText(QString::number(f.pointSize())));
    actionTextBold->setChecked(f.bold());
    actionTextItalic->setChecked(f.italic());
    actionTextUnderline->setChecked(f.underline());
}

void TextEdit::colorChanged(const QColor &c)
{
    QPixmap pix(16, 16);
    pix.fill(c);
    actionTextColor->setIcon(pix);
}

void TextEdit::alignmentChanged(Qt::Alignment a)
{
    if (a & Qt::AlignLeft)
        actionAlignLeft->setChecked(true);
    else if (a & Qt::AlignHCenter)
        actionAlignCenter->setChecked(true);
    else if (a & Qt::AlignRight)
        actionAlignRight->setChecked(true);
    else if (a & Qt::AlignJustify)
        actionAlignJustify->setChecked(true);
}
```

Файл «odt_writer.h»

```
#ifndef ODT_WRITER_H
#define ODT_WRITER_H

#include <QCoreApplication>
#include <QHash>
#include <QString>
#include <QXmlStreamWriter>
class QTextBlockFormat;
class QTextCharFormat;
class QTextDocument;

class OdtWriter
{
    Q_DECLARE_TR_FUNCTIONS(OdtWriter)

public:
    OdtWriter();

    QString errorString() const
```

```
    {
        return m_error;
    }

    void setFlatXML(bool flat);

    bool write(QIODevice* device, const QTextDocument* document);

private:
    bool writeCompressed(QIODevice* device, const QTextDocument* document);
    bool writeUncompressed(QIODevice* device, const QTextDocument* document);
    QByteArray writeDocument(const QTextDocument* document);
    QByteArray writeStylesDocument(const QTextDocument* document);
    void writeStyles(const QTextDocument* document);
    void writeAutomaticStyles(const QTextDocument* document);
    bool writeParagraphStyle(const QTextBlockFormat& format, const QString& name);
    bool writeTextStyle(const QTextCharFormat& format, const QString& name);
    void writeBody(const QTextDocument* document);

private:
    QXmlStreamWriter m_xml;
    QHash<int, QString> m_styles;
    QString m_error;
    bool m_flat;
};

#endif
```

Файл «odt_writer.cpp»

```
#include "odt_writer.h"
#include "qtzipwriter.h"
#include <QBuffer>
#include <QTextBlock>
#include <QTextBlockFormat>
#include <QTextCharFormat>
#include <QTextDocument>
#include <QTextFragment>

OdtWriter::OdtWriter() :
    m_flat(false)
{
}

//-----------------------------------------------------------------------------

void OdtWriter::setFlatXML(bool flat)
{
    m_flat = flat;
}

//-----------------------------------------------------------------------------

bool OdtWriter::write(QIODevice* device, const QTextDocument* document)
{
    if (!m_flat) {
        return writeCompressed(device, document);
    } else {
        return writeUncompressed(device, document);
    }
}

//-----------------------------------------------------------------------------
```

```
bool OdtWriter::writeCompressed(QIODevice* device, const QTextDocument* document)
{
    QtZipWriter zip(device);
    if (zip.status() != QtZipWriter::NoError) {
        return false;
    }

    zip.setCompressionPolicy(QtZipWriter::NeverCompress);
    zip.addFile(QString::fromLatin1("mimetype"),
        "application/vnd.oasis.opendocument.text");
    zip.setCompressionPolicy(QtZipWriter::AlwaysCompress);

    zip.addFile(QString::fromLatin1("META-INF/manifest.xml"),
        "<?xml version=\"1.0\" encoding=\"UTF-8\"?>\n"
        "<manifest:manifest
xmlns:manifest=\"urn:oasis:names:tc:opendocument:xmlns:manifest:1.0\"
manifest:version=\"1.2\">\n"
        " <manifest:file-entry manifest:full-path=\"/\" manifest:version=\"1.2\"
manifest:media-type=\"application/vnd.oasis.opendocument.text\"/>\n"
        " <manifest:file-entry manifest:full-path=\"content.xml\" manifest:media-
type=\"text/xml\"/>\n"
        " <manifest:file-entry manifest:full-path=\"styles.xml\" manifest:media-
type=\"text/xml\"/>\n"
        "</manifest:manifest>\n");

    zip.addFile(QString::fromLatin1("content.xml"),
        writeDocument(document));

    zip.addFile(QString::fromLatin1("styles.xml"),
        writeStylesDocument(document));

    zip.close();

//    return zip.status() == QtZipWriter::NoError;
}

//-----------------------------------------------------------------------------

bool OdtWriter::writeUncompressed(QIODevice* device, const QTextDocument* document)
{
    m_xml.setDevice(device);
    m_xml.setCodec("UTF-8");
    m_xml.setAutoFormatting(true);
    m_xml.setAutoFormattingIndent(1);


m_xml.writeNamespace(QString::fromLatin1("urn:oasis:names:tc:opendocument:xmlns:offic
e:1.0"), QString::fromLatin1("office"));

m_xml.writeNamespace(QString::fromLatin1("urn:oasis:names:tc:opendocument:xmlns:style
:1.0"), QString::fromLatin1("style"));

m_xml.writeNamespace(QString::fromLatin1("urn:oasis:names:tc:opendocument:xmlns:text:
1.0"), QString::fromLatin1("text"));

m_xml.writeNamespace(QString::fromLatin1("urn:oasis:names:tc:opendocument:xmlns:xsl-
fo-compatible:1.0"), QString::fromLatin1("fo"));

    m_xml.writeStartDocument();
    m_xml.writeStartElement(QString::fromLatin1("office:document"));
    m_xml.writeAttribute(QString::fromLatin1("office:mimetype"),
QString::fromLatin1("application/vnd.oasis.opendocument.text"));
```

```
    m_xml.writeAttribute(QString::fromLatin1("office:version"),
QString::fromLatin1("1.2"));

    writeStyles(document);
    writeAutomaticStyles(document);
    writeBody(document);

    m_xml.writeEndElement();
    m_xml.writeEndDocument();

    return !m_xml.hasError();
}

//------------------------------------------------------------------------

QByteArray OdtWriter::writeDocument(const QTextDocument* document)
{
    QByteArray data;
    QBuffer buffer(&data);
    buffer.open(QIODevice::WriteOnly);

    m_xml.setDevice(&buffer);
    m_xml.setCodec("UTF-8");
    m_xml.setAutoFormatting(true);
    m_xml.setAutoFormattingIndent(1);


m_xml.writeNamespace(QString::fromLatin1("urn:oasis:names:tc:opendocument:xmlns:offic
e:1.0"), QString::fromLatin1("office"));

m_xml.writeNamespace(QString::fromLatin1("urn:oasis:names:tc:opendocument:xmlns:style
:1.0"), QString::fromLatin1("style"));

m_xml.writeNamespace(QString::fromLatin1("urn:oasis:names:tc:opendocument:xmlns:text:
1.0"), QString::fromLatin1("text"));

m_xml.writeNamespace(QString::fromLatin1("urn:oasis:names:tc:opendocument:xmlns:xsl-
fo-compatible:1.0"), QString::fromLatin1("fo"));

    m_xml.writeStartDocument();
    m_xml.writeStartElement(QString::fromLatin1("office:document-content"));
    m_xml.writeAttribute(QString::fromLatin1("office:version"),
QString::fromLatin1("1.2"));

    writeAutomaticStyles(document);
    writeBody(document);

    m_xml.writeEndElement();
    m_xml.writeEndDocument();

    buffer.close();
    return data;
}

//------------------------------------------------------------------------

QByteArray OdtWriter::writeStylesDocument(const QTextDocument* document)
{
    QByteArray data;
    QBuffer buffer(&data);
    buffer.open(QIODevice::WriteOnly);

    m_xml.setDevice(&buffer);
```

```
    m_xml.setCodec("UTF-8");
    m_xml.setAutoFormatting(true);
    m_xml.setAutoFormattingIndent(1);


m_xml.writeNamespace(QString::fromLatin1("urn:oasis:names:tc:opendocument:xmlns:office:1.0"), QString::fromLatin1("office"));

m_xml.writeNamespace(QString::fromLatin1("urn:oasis:names:tc:opendocument:xmlns:style:1.0"), QString::fromLatin1("style"));

m_xml.writeNamespace(QString::fromLatin1("urn:oasis:names:tc:opendocument:xmlns:xsl-fo-compatible:1.0"), QString::fromLatin1("fo"));

    m_xml.writeStartDocument();
    m_xml.writeStartElement(QString::fromLatin1("office:document-styles"));
    m_xml.writeAttribute(QString::fromLatin1("office:version"),
QString::fromLatin1("1.2"));

    writeStyles(document);

    m_xml.writeEndElement();
    m_xml.writeEndDocument();

    buffer.close();
    return data;
}

//------------------------------------------------------------------------------

void OdtWriter::writeStyles(const QTextDocument*)
{
    static const std::vector<std::vector<QString>> styles = {
        {"Normal", "Normal", "0", "12pt", "normal"},
        {"Heading-1", "Heading 1", "1", "18pt", "bold"},
        {"Heading-2", "Heading 2", "2", "16pt", "bold"},
        {"Heading-3", "Heading 3", "3", "14pt", "bold"},
        {"Heading-4", "Heading 4", "4", "12pt", "bold"},
        {"Heading-5", "Heading 5", "5", "10pt", "bold"},
        {"Heading-6", "Heading 6", "6", "8pt", "bold"}
    };

    m_xml.writeStartElement("office:styles");
    for (const auto& style : styles) {
        m_xml.writeStartElement("style:style");
        m_xml.writeAttribute("style:name", style[0]);
        m_xml.writeAttribute("style:display-name", style[1]);
        m_xml.writeAttribute("style:family", "paragraph");
        if (style[0] != "Normal") {
            m_xml.writeAttribute("style:parent-style-name", "Normal");
            m_xml.writeAttribute("style:next-style-name", "Normal");
            m_xml.writeAttribute("style:default-outline-level", style[2]);
        }

        m_xml.writeStartElement("style:text-properties");
        m_xml.writeAttribute("fo:font-size", style[3]);
        m_xml.writeAttribute("fo:font-weight", style[4]);
        m_xml.writeEndElement();

        m_xml.writeEndElement();
    }
    m_xml.writeEndElement();
}
```

```
//-----------------------------------------------------------------------------

void OdtWriter::writeAutomaticStyles(const QTextDocument* document)
{
    m_xml.writeStartElement(QString::fromLatin1("office:automatic-styles"));

    QVector<QTextFormat> formats = document->allFormats();

    // Find all used styles
    QVector<int> text_styles;
    QVector<int> paragraph_styles;
    int index = 0;
    for (QTextBlock block = document->begin(); block.isValid(); block = block.next())
{
        index = block.blockFormatIndex();
        if (!paragraph_styles.contains(index)) {
            int heading =
block.blockFormat().property(QTextFormat::UserProperty).toInt();
            if (!heading) {
                paragraph_styles.append(index);
            } else {
                m_styles.insert(index, QString("Heading-%1").arg(heading));
            }
        }
        for (QTextBlock::iterator iter = block.begin(); !(iter.atEnd()); ++iter) {
            index = iter.fragment().charFormatIndex();
            if (!text_styles.contains(index) && formats.at(index).propertyCount()) {
                text_styles.append(index);
            }
        }
    }

    // Write text styles
    int text_style = 1;
    for (int i = 0; i < text_styles.size(); ++i) {
        int index = text_styles.at(i);
        const QTextFormat& format = formats.at(index);
        QString name = QString::fromLatin1("T") + QString::number(text_style);
        if (writeTextStyle(format.toCharFormat(), name)) {
            m_styles.insert(index, name);
            ++text_style;
        }
    }

    // Write paragraph styles
    int paragraph_style = 1;
    for (int i = 0; i < paragraph_styles.size(); ++i) {
        int index = paragraph_styles.at(i);
        const QTextFormat& format = formats.at(index);
        QString name = QString::fromLatin1("P") + QString::number(paragraph_style);
        if (writeParagraphStyle(format.toBlockFormat(), name)) {
            m_styles.insert(index, name);
            ++paragraph_style;
        } else {
            m_styles.insert(index, "Normal");
        }
    }

    m_xml.writeEndElement();
}

//-----------------------------------------------------------------------------
```

```
bool OdtWriter::writeParagraphStyle(const QTextBlockFormat& format, const QString&
name)
{
    QXmlStreamAttributes attributes;
    bool rtl = format.layoutDirection() == Qt::RightToLeft;
    if (rtl) {
        attributes.append(QString::fromLatin1("style:writing-mode"),
QString::fromLatin1("rl"));
    }

    Qt::Alignment align = format.alignment();
    if (rtl && (align & Qt::AlignLeft)) {
        attributes.append(QString::fromLatin1("fo:text-align"),
QString::fromLatin1("left"));
    } else if (align & Qt::AlignRight) {
        attributes.append(QString::fromLatin1("fo:text-align"),
QString::fromLatin1("right"));
    } else if (align & Qt::AlignCenter) {
        attributes.append(QString::fromLatin1("fo:text-align"),
QString::fromLatin1("center"));
    } else if (align & Qt::AlignJustify) {
        attributes.append(QString::fromLatin1("fo:text-align"),
QString::fromLatin1("justify"));
    }

    if (format.indent() > 0) {
        attributes.append(QString::fromLatin1("fo:margin-left"),
QString::number(format.indent() * 0.5) + QString::fromLatin1("in"));
    }

    if (attributes.isEmpty()) {
        return false;
    }

    m_xml.writeStartElement(QString::fromLatin1("style:style"));
    m_xml.writeAttribute(QString::fromLatin1("style:name"), name);
    m_xml.writeAttribute(QString::fromLatin1("style:family"),
QString::fromLatin1("paragraph"));
    m_xml.writeAttribute(QString::fromLatin1("style:parent-style-name"),
QString::fromLatin1("Normal"));

    m_xml.writeEmptyElement(QString::fromLatin1("style:paragraph-properties"));
    m_xml.writeAttributes(attributes);
    m_xml.writeEndElement();

    return true;
}

//---------------------------------------------------------------------------

bool OdtWriter::writeTextStyle(const QTextCharFormat& format, const QString& name)
{
    QXmlStreamAttributes attributes;
    if (format.fontWeight() == QFont::Bold) {
        attributes.append(QString::fromLatin1("fo:font-weight"),
QString::fromLatin1("bold"));
    }
    if (format.fontItalic()) {
        attributes.append(QString::fromLatin1("fo:font-style"),
QString::fromLatin1("italic"));
    }
    if (format.fontUnderline()) {
```

```
        attributes.append(QString::fromLatin1("style:text-underline-type"),
QString::fromLatin1("single"));
        attributes.append(QString::fromLatin1("style:text-underline-style"),
QString::fromLatin1("solid"));
    }
    if (format.fontStrikeOut()) {
        attributes.append(QString::fromLatin1("style:text-line-through-type"),
QString::fromLatin1("single"));
    }
    if (format.verticalAlignment() == QTextCharFormat::AlignSuperScript) {
        attributes.append(QString::fromLatin1("style:text-position"),
QString::fromLatin1("super"));
    } else if (format.verticalAlignment() == QTextCharFormat::AlignSubScript) {
        attributes.append(QString::fromLatin1("style:text-position"),
QString::fromLatin1("sub"));
    }

    if (attributes.isEmpty()) {
        return false;
    }

    m_xml.writeStartElement(QString::fromLatin1("style:style"));
    m_xml.writeAttribute(QString::fromLatin1("style:name"), name);
    m_xml.writeAttribute(QString::fromLatin1("style:family"),
QString::fromLatin1("text"));

    m_xml.writeEmptyElement(QString::fromLatin1("style:text-properties"));
    m_xml.writeAttributes(attributes);
    m_xml.writeEndElement();

    return true;
}

//---------------------------------------------------------------------------

void OdtWriter::writeBody(const QTextDocument* document)
{
    m_xml.writeStartElement(QString::fromLatin1("office:body"));
    m_xml.writeStartElement(QString::fromLatin1("office:text"));

    for (QTextBlock block = document->begin(); block.isValid(); block = block.next())
{
        int heading =
block.blockFormat().property(QTextFormat::UserProperty).toInt();
        if (!heading) {
            m_xml.writeStartElement(QString::fromLatin1("text:p"));
        } else {
            m_xml.writeStartElement(QString::fromLatin1("text:h"));
            m_xml.writeAttribute(QString::fromLatin1("text:outline-level"),
QString::number(heading));
        }
        m_xml.writeAttribute(QString::fromLatin1("text:style-name"),
m_styles.value(block.blockFormatIndex()));
        m_xml.setAutoFormatting(false);

        for (QTextBlock::iterator iter = block.begin(); !(iter.atEnd()); ++iter) {
            QTextFragment fragment = iter.fragment();
            QString style = m_styles.value(fragment.charFormatIndex());
            if (!style.isEmpty()) {
                m_xml.writeStartElement(QString::fromLatin1("text:span"));
                m_xml.writeAttribute(QString::fromLatin1("text:style-name"), style);
            }
```

```
            QString text = fragment.text();
            int start = 0;
            int spaces = -1;
            for (int i = 0; i < text.length(); ++i) {
                QChar c = text.at(i);
                if (c.unicode() == 0x0) {
                    m_xml.writeCharacters(text.mid(start, i - start));
                    start = i + 1;
                } else if (c.unicode() == 0x0009) {
                    m_xml.writeCharacters(text.mid(start, i - start));
                    m_xml.writeEmptyElement(QString::fromLatin1("text:tab"));
                    start = i + 1;
                } else if (c.unicode() == 0x2028) {
                    m_xml.writeCharacters(text.mid(start, i - start));
                    m_xml.writeEmptyElement(QString::fromLatin1("text:line-break"));
                    start = i + 1;
                } else if (c.unicode() == 0x0020) {
                    ++spaces;
                } else if (spaces > 0) {
                    m_xml.writeCharacters(text.mid(start, i - spaces - start));
                    m_xml.writeEmptyElement(QString::fromLatin1("text:s"));
                    m_xml.writeAttribute(QString::fromLatin1("text:c"),
QString::number(spaces));
                    spaces = -1;
                    start = i;
                } else {
                    spaces = -1;
                }
            }
            if (spaces > 0) {
                m_xml.writeCharacters(text.mid(start, text.length() - spaces -
start));
                m_xml.writeEmptyElement(QString::fromLatin1("text:s"));
                m_xml.writeAttribute(QString::fromLatin1("text:c"),
QString::number(spaces));
            } else {
                m_xml.writeCharacters(text.mid(start));
            }

            if (!style.isEmpty()) {
                m_xml.writeEndElement();
            }
        }

        m_xml.writeEndElement();
        m_xml.setAutoFormatting(true);
    }

    m_xml.writeEndElement();
    m_xml.writeEndElement();
}
```

Файл «format_reader.h»

```
#ifndef FORMAT_READER_H
#define FORMAT_READER_H

#include <QString>
#include <QTextCursor>
class QIODevice;
class QTextDocument;

class FormatReader
```

```
{
public:
    virtual ~FormatReader()
    {
    }

    QByteArray encoding() const
    {
        return m_encoding;
    }

    QString errorString() const
    {
        return m_error;
    }

    bool hasError() const
    {
        return !m_error.isEmpty();
    }

    void read(QIODevice* device, QTextDocument* document)
    {
        m_cursor = QTextCursor(document);
        readData(device);
    }

    void read(QIODevice* device, const QTextCursor& cursor)
    {
        m_cursor = cursor;
        readData(device);
    }

    enum { Type = 0 };
    virtual int type() const
    {
        return Type;
    }

protected:
    QTextCursor m_cursor;
    QString m_error;
    QByteArray m_encoding;

private:
    virtual void readData(QIODevice* device) = 0;
};

#endif
```

Файл «qtzipwriter.h»

```
#ifndef QTZIPWRITER_H
#define QTZIPWRITER_H

#include <QtGlobal>

#include <QFile>
#include <QString>

class QtZipWriterPrivate;

class QtZipWriter
{
```

```
public:
    explicit QtZipWriter(const QString &fileName, QIODevice::OpenMode mode =
(QIODevice::WriteOnly | QIODevice::Truncate) );

    explicit QtZipWriter(QIODevice *device);
    ~QtZipWriter();

    QIODevice* device() const;

    bool isWritable() const;
    bool exists() const;

    enum Status {
        NoError,
        FileWriteError,
        FileOpenError,
        FilePermissionsError,
        FileError
    };

    Status status() const;

    enum CompressionPolicy {
        AlwaysCompress,
        NeverCompress,
        AutoCompress
    };

    void setCompressionPolicy(CompressionPolicy policy);
    CompressionPolicy compressionPolicy() const;

    void setCreationPermissions(QFile::Permissions permissions);
    QFile::Permissions creationPermissions() const;

    void addFile(const QString &fileName, const QByteArray &data);

    void addFile(const QString &fileName, QIODevice *device);

    void addDirectory(const QString &dirName);

    void addSymLink(const QString &fileName, const QString &destination);

    void close();
private:
    QtZipWriterPrivate *d;
    Q_DISABLE_COPY(QtZipWriter)
};

#endif // QTZIPWRITER_H
```

Файл «txtreader.h»

```
#ifndef TXT_READER_H
#define TXT_READER_H

#include "format_reader.h"

class TxtReader : public FormatReader
{
public:
    TxtReader();

    enum { Type = 1 };
    int type() const
```

```
    {
        return Type;
    }

    static bool canRead(QIODevice* device)
    {
        Q_UNUSED(device)
        return true;
    }

private:
    void readData(QIODevice* device);
};

#endif
```

Файл «qtzipwirter.cpp»

```
#include "qtzipwriter.h"
#include <QDateTime>
#include <QDir>
#include <QtDebug>
#include <QtEndian>
#include <QtGlobal>

#include <zlib.h>

#define ZIP_VERSION 20

#if 0
#define ZDEBUG qDebug
#else
#define ZDEBUG if (0) qDebug
#endif

static inline uint readUInt(const uchar *data)
{
    return (data[0]) + (data[1]<<8) + (data[2]<<16) + (data[3]<<24);
}

static inline ushort readUShort(const uchar *data)
{
    return (data[0]) + (data[1]<<8);
}

static inline void writeUInt(uchar *data, uint i)
{
    data[0] = i & 0xff;
    data[1] = (i>>8) & 0xff;
    data[2] = (i>>16) & 0xff;
    data[3] = (i>>24) & 0xff;
}

static inline void writeUShort(uchar *data, ushort i)
{
    data[0] = i & 0xff;
    data[1] = (i>>8) & 0xff;
}

static inline void copyUInt(uchar *dest, const uchar *src)
{
    dest[0] = src[0];
    dest[1] = src[1];
    dest[2] = src[2];
```

```
    dest[3] = src[3];
}

static inline void copyUShort(uchar *dest, const uchar *src)
{
    dest[0] = src[0];
    dest[1] = src[1];
}

static void writeMSDosDate(uchar *dest, const QDateTime& dt)
{
    if (dt.isValid()) {
        quint16 time =
            (dt.time().hour() << 11)    // 5 bit hour
            | (dt.time().minute() << 5)   // 6 bit minute
            | (dt.time().second() >> 1);  // 5 bit double seconds

        dest[0] = time & 0xff;
        dest[1] = time >> 8;

        quint16 date =
            ((dt.date().year() - 1980) << 9) // 7 bit year 1980-based
            | (dt.date().month() << 5)           // 4 bit month
            | (dt.date().day());                 // 5 bit day

        dest[2] = char(date);
        dest[3] = char(date >> 8);
    } else {
        dest[0] = 0;
        dest[1] = 0;
        dest[2] = 0;
        dest[3] = 0;
    }
}

static int inflate(Bytef *dest, ulong *destLen, const Bytef *source, ulong sourceLen)
{
    z_stream stream;
    int err;

    stream.next_in = const_cast<Bytef*>(source);
    stream.avail_in = (uInt)sourceLen;
    if ((uLong)stream.avail_in != sourceLen)
        return Z_BUF_ERROR;

    stream.next_out = dest;
    stream.avail_out = (uInt)*destLen;
    if ((uLong)stream.avail_out != *destLen)
        return Z_BUF_ERROR;

    stream.zalloc = (alloc_func)0;
    stream.zfree = (free_func)0;

    if (err != Z_OK)
        return err;

    if (err != Z_STREAM_END) {
        if (err == Z_NEED_DICT || (err == Z_BUF_ERROR && stream.avail_in == 0))
            return Z_DATA_ERROR;
        return err;
    }
    *destLen = stream.total_out;
    return err;
```

```
}

static int deflate (Bytef *dest, ulong *destLen, const Bytef *source, ulong
sourceLen)
{
    z_stream stream;
    int err;

    stream.next_in = const_cast<Bytef*>(source);
    stream.avail_in = (uInt)sourceLen;
    stream.next_out = dest;
    stream.avail_out = (uInt)*destLen;
    if ((uLong)stream.avail_out != *destLen) return Z_BUF_ERROR;

    stream.zalloc = (alloc_func)0;
    stream.zfree = (free_func)0;
    stream.opaque = (voidpf)0;
    if (err != Z_OK) return err;

    if (err != Z_STREAM_END) {
        return err == Z_OK ? Z_BUF_ERROR : err;
    }
    *destLen = stream.total_out;

    return err;
}


namespace WindowsFileAttributes {
enum {
    Dir         = 0x10, // FILE_ATTRIBUTE_DIRECTORY
    File        = 0x80, // FILE_ATTRIBUTE_NORMAL
    TypeMask    = 0x90,

    ReadOnly    = 0x01, // FILE_ATTRIBUTE_READONLY
    PermMask    = 0x01
};
}

namespace UnixFileAttributes {
enum {
    Dir         = 0040000, // __S_IFDIR
    File        = 0100000, // __S_IFREG
    SymLink     = 0120000, // __S_IFLNK
    TypeMask    = 0170000, // __S_IFMT

    ReadUser    = 0400, // __S_IRUSR
    WriteUser   = 0200, // __S_IWUSR
    ExeUser     = 0100, // __S_IXUSR
    ReadGroup   = 0040, // __S_IRGRP
    WriteGroup  = 0020, // __S_IWGRP
    ExeGroup    = 0010, // __S_IXGRP
    ReadOther   = 0004, // __S_IROTH
    WriteOther  = 0002, // __S_IWOTH
    ExeOther    = 0001, // __S_IXOTH
    PermMask    = 0777
};
}

static QFile::Permissions modeToPermissions(quint32 mode)
{
    QFile::Permissions ret;
    if (mode & UnixFileAttributes::ReadUser)
```

```
        ret |= QFile::ReadOwner | QFile::ReadUser;
    if (mode & UnixFileAttributes::WriteUser)
        ret |= QFile::WriteOwner | QFile::WriteUser;
    if (mode & UnixFileAttributes::ExeUser)
        ret |= QFile::ExeOwner | QFile::ExeUser;
    if (mode & UnixFileAttributes::ReadGroup)
        ret |= QFile::ReadGroup;
    if (mode & UnixFileAttributes::WriteGroup)
        ret |= QFile::WriteGroup;
    if (mode & UnixFileAttributes::ExeGroup)
        ret |= QFile::ExeGroup;
    if (mode & UnixFileAttributes::ReadOther)
        ret |= QFile::ReadOther;
    if (mode & UnixFileAttributes::WriteOther)
        ret |= QFile::WriteOther;
    if (mode & UnixFileAttributes::ExeOther)
        ret |= QFile::ExeOther;
    return ret;
}

static quint32 permissionsToMode(QFile::Permissions perms)
{
    quint32 mode = 0;
    if (perms & (QFile::ReadOwner | QFile::ReadUser))
        mode |= UnixFileAttributes::ReadUser;
    if (perms & (QFile::WriteOwner | QFile::WriteUser))
        mode |= UnixFileAttributes::WriteUser;
    if (perms & (QFile::ExeOwner | QFile::ExeUser))
        mode |= UnixFileAttributes::WriteUser;
    if (perms & QFile::ReadGroup)
        mode |= UnixFileAttributes::ReadGroup;
    if (perms & QFile::WriteGroup)
        mode |= UnixFileAttributes::WriteGroup;
    if (perms & QFile::ExeGroup)
        mode |= UnixFileAttributes::ExeGroup;
    if (perms & QFile::ReadOther)
        mode |= UnixFileAttributes::ReadOther;
    if (perms & QFile::WriteOther)
        mode |= UnixFileAttributes::WriteOther;
    if (perms & QFile::ExeOther)
        mode |= UnixFileAttributes::ExeOther;
    return mode;
}

static QDateTime readMSDosDate(const uchar *src)
{
    uint dosDate = readUInt(src);
    quint64 uDate;
    uDate = (quint64)(dosDate >> 16);
    uint tm_mday = (uDate & 0x1f);
    uint tm_mon =  ((uDate & 0x1E0) >> 5);
    uint tm_year = (((uDate & 0x0FE00) >> 9) + 1980);
    uint tm_hour = ((dosDate & 0xF800) >> 11);
    uint tm_min =  ((dosDate & 0x7E0) >> 5);
    uint tm_sec =  ((dosDate & 0x1f) << 1);

    return QDateTime(QDate(tm_year, tm_mon, tm_mday), QTime(tm_hour, tm_min,
tm_sec));
}

// for details, see http://www.pkware.com/documents/casestudies/APPNOTE.TXT

namespace
```

```
{

enum HostOS {
    HostFAT      = 0,
    HostAMIGA    = 1,
    HostVMS      = 2,  // VAX/VMS
    HostUnix     = 3,
    HostVM_CMS   = 4,
    HostAtari    = 5,  // what if it's a minix filesystem? [cjh]
    HostHPFS     = 6,  // filesystem used by OS/2 (and NT 3.x)
    HostMac      = 7,
    HostZ_System = 8,
    HostCPM      = 9,
    HostTOPS20   = 10, // pkzip 2.50 NTFS
    HostNTFS     = 11, // filesystem used by Windows NT
    HostQDOS     = 12, // SMS/QDOS
    HostAcorn    = 13, // Archimedes Acorn RISC OS
    HostVFAT     = 14, // filesystem used by Windows 95, NT
    HostMVS      = 15,
    HostBeOS     = 16, // hybrid POSIX/database filesystem
    HostTandem   = 17,
    HostOS400    = 18,
    HostOSX      = 19
};

enum GeneralPurposeFlag {
    Encrypted = 0x01,
    AlgTune1 = 0x02,
    AlgTune2 = 0x04,
    HasDataDescriptor = 0x08,
    PatchedData = 0x20,
    StrongEncrypted = 0x40,
    Utf8Names = 0x0800,
    CentralDirectoryEncrypted = 0x2000
};

enum CompressionMethod {
    CompressionMethodStored = 0,
    CompressionMethodShrunk = 1,
    CompressionMethodReduced1 = 2,
    CompressionMethodReduced2 = 3,
    CompressionMethodReduced3 = 4,
    CompressionMethodReduced4 = 5,
    CompressionMethodImploded = 6,
    CompressionMethodReservedTokenizing = 7, // reserved for tokenizing
    CompressionMethodDeflated = 8,
    CompressionMethodDeflated64 = 9,
    CompressionMethodPKImploding = 10,

    CompressionMethodBZip2 = 12,

    CompressionMethodLZMA = 14,

    CompressionMethodTerse = 18,
    CompressionMethodLz77 = 19,

    CompressionMethodJpeg = 96,
    CompressionMethodWavPack = 97,
    CompressionMethodPPMd = 98,
    CompressionMethodWzAES = 99
};

struct LocalFileHeader
```

```
{
    uchar signature[4]; //  0x04034b50
    uchar version_needed[2];
    uchar general_purpose_bits[2];
    uchar compression_method[2];
    uchar last_mod_file[4];
    uchar crc_32[4];
    uchar compressed_size[4];
    uchar uncompressed_size[4];
    uchar file_name_length[2];
    uchar extra_field_length[2];
};

struct DataDescriptor
{
    uchar crc_32[4];
    uchar compressed_size[4];
    uchar uncompressed_size[4];
};

struct CentralFileHeader
{
    uchar signature[4]; // 0x02014b50
    uchar version_made[2];
    uchar version_needed[2];
    uchar general_purpose_bits[2];
    uchar compression_method[2];
    uchar last_mod_file[4];
    uchar crc_32[4];
    uchar compressed_size[4];
    uchar uncompressed_size[4];
    uchar file_name_length[2];
    uchar extra_field_length[2];
    uchar file_comment_length[2];
    uchar disk_start[2];
    uchar internal_file_attributes[2];
    uchar external_file_attributes[4];
    uchar offset_local_header[4];
    LocalFileHeader toLocalHeader() const;
};

struct EndOfDirectory
{
    uchar signature[4]; // 0x06054b50
    uchar this_disk[2];
    uchar start_of_directory_disk[2];
    uchar num_dir_entries_this_disk[2];
    uchar num_dir_entries[2];
    uchar directory_size[4];
    uchar dir_start_offset[4];
    uchar comment_length[2];
};

struct FileHeader
{
    CentralFileHeader h;
    QByteArray file_name;
    QByteArray extra_field;
    QByteArray file_comment;
};

}
```

```
class QtZipPrivate
{
public:
    QtZipPrivate(QIODevice *device, bool ownDev)
        : device(device), ownDevice(ownDev), dirtyFileTree(true),
start_of_directory(0)
    {
    }

    ~QtZipPrivate()
    {
        if (ownDevice)
            delete device;
    }

    QIODevice *device;
    bool ownDevice;
    bool dirtyFileTree;
    QVector<FileHeader> fileHeaders;
    QByteArray comment;
    uint start_of_directory;
};

class QtZipWriterPrivate : public QtZipPrivate
{
public:
    QtZipWriterPrivate(QIODevice *device, bool ownDev)
        : QtZipPrivate(device, ownDev),
        status(QtZipWriter::NoError),
        permissions(QFile::ReadOwner | QFile::WriteOwner),
        compressionPolicy(QtZipWriter::AlwaysCompress)
    {
    }

    QtZipWriter::Status status;
    QFile::Permissions permissions;
    QtZipWriter::CompressionPolicy compressionPolicy;

    enum EntryType { Directory, File, Symlink };

    void addEntry(EntryType type, const QString &fileName, const QByteArray
&contents);
};

LocalFileHeader CentralFileHeader::toLocalHeader() const
{
    LocalFileHeader h;
    writeUInt(h.signature, 0x04034b50);
    copyUShort(h.version_needed, version_needed);
    copyUShort(h.general_purpose_bits, general_purpose_bits);
    copyUShort(h.compression_method, compression_method);
    copyUInt(h.last_mod_file, last_mod_file);
    copyUInt(h.crc_32, crc_32);
    copyUInt(h.compressed_size, compressed_size);
    copyUInt(h.uncompressed_size, uncompressed_size);
    copyUShort(h.file_name_length, file_name_length);
    copyUShort(h.extra_field_length, extra_field_length);
    return h;
}

void QtZipWriterPrivate::addEntry(EntryType type, const QString &fileName, const
QByteArray &contents/*, QFile::Permissions permissions, QtZip::Method m*/)
{
```

```
#ifndef NDEBUG
    static const char *const entryTypes[] = {
        "directory",
        "file     ",
        "symlink  " };
    ZDEBUG() << "adding" << entryTypes[type] <<":" << fileName.toUtf8().data() <<
(type == 2 ? QByteArray(" -> " + contents).constData() : "");
#endif

    if (! (device->isOpen() || device->open(QIODevice::WriteOnly))) {
        status = QtZipWriter::FileOpenError;
        return;
    }
    device->seek(start_of_directory);

    // don't compress small files
    QtZipWriter::CompressionPolicy compression = compressionPolicy;
    if (compressionPolicy == QtZipWriter::AutoCompress) {
        if (contents.length() < 64)
            compression = QtZipWriter::NeverCompress;
        else
            compression = QtZipWriter::AlwaysCompress;
    }

    FileHeader header;
    memset(&header.h, 0, sizeof(CentralFileHeader));
    writeUInt(header.h.signature, 0x02014b50);

    writeUShort(header.h.version_needed, ZIP_VERSION);
    writeUInt(header.h.uncompressed_size, contents.length());
    writeMSDosDate(header.h.last_mod_file, QDateTime::currentDateTime());
    QByteArray data = contents;
    if (compression == QtZipWriter::AlwaysCompress) {
        writeUShort(header.h.compression_method, CompressionMethodDeflated);

       ulong len = contents.length();
        // shamelessly copied form zlib
        len += (len >> 12) + (len >> 14) + 11;
        int res;
        do {
            data.resize(len);
            res = deflate((uchar*)data.data(), &len, (const
uchar*)contents.constData(), contents.length());

            switch (res) {
            case Z_OK:
                data.resize(len);
                break;
            case Z_MEM_ERROR:
                qWarning("QtZip: Z_MEM_ERROR: Not enough memory to compress file,
skipping");
                data.resize(0);
                break;
            case Z_BUF_ERROR:
                len *= 2;
                break;
            }
        } while (res == Z_BUF_ERROR);
    }
// TODO add a check if data.length() > contents.length().  Then try to store the
original and revert the compression method to be uncompressed
    writeUInt(header.h.compressed_size, data.length());
```

```
    // if bit 11 is set, the filename and comment fields must be encoded using UTF-8
    ushort general_purpose_bits = Utf8Names; // always use utf-8
    writeUShort(header.h.general_purpose_bits, general_purpose_bits);

    const bool inUtf8 = (general_purpose_bits & Utf8Names) != 0;
    header.file_name = inUtf8 ? fileName.toUtf8() : fileName.toLocal8Bit();
    if (header.file_name.size() > 0xffff) {
        qWarning("QtZip: Filename is too long, chopping it to 65535 bytes");
        header.file_name = header.file_name.left(0xffff); // ### don't break the utf-
8 sequence, if any
    }
    if (header.file_comment.size() + header.file_name.size() > 0xffff) {
        qWarning("QtZip: File comment is too long, chopping it to 65535 bytes");
        header.file_comment.truncate(0xffff - header.file_name.size()); // ### don't
break the utf-8 sequence, if any
    }
    writeUShort(header.h.file_name_length, header.file_name.length());
    //h.extra_field_length[2];

    writeUShort(header.h.version_made, HostUnix << 8);
    //uchar internal_file_attributes[2];
    //uchar external_file_attributes[4];
    quint32 mode = permissionsToMode(permissions);
    switch (type) {
    case Symlink:
        mode |= UnixFileAttributes::SymLink;
        break;
    case Directory:
        mode |= UnixFileAttributes::Dir;
        break;
    case File:
        mode |= UnixFileAttributes::File;
        break;
    default:
        Q_UNREACHABLE();
        break;
    }
    writeUInt(header.h.external_file_attributes, mode << 16);
    writeUInt(header.h.offset_local_header, start_of_directory);


    fileHeaders.append(header);

    LocalFileHeader h = header.h.toLocalHeader();
    device->write((const char *)&h, sizeof(LocalFileHeader));
    device->write(header.file_name);
    device->write(data);
    start_of_directory = device->pos();
    dirtyFileTree = true;
}

/*!
    \class QtZipWriter
    \internal
    \since 4.5
    \brief the QtZipWriter class provides a way to create a new zip archive.
    QtZipWriter can be used to create a zip archive containing any number of files
    and directories. The files in the archive will be compressed in a way that is
    compatible with common zip reader applications.
*/


/*!
```

```
    Create a new zip archive that operates on the \a archive filename.  The file will
    be opened with the \a mode.
    \sa isValid()
*/
QtZipWriter::QtZipWriter(const QString &fileName, QIODevice::OpenMode mode)
{
    QScopedPointer<QFile> f(new QFile(fileName));
    QtZipWriter::Status status;
    if (f->open(mode) && f->error() == QFile::NoError)
        status = QtZipWriter::NoError;
    else {
        if (f->error() == QFile::WriteError)
            status = QtZipWriter::FileWriteError;
        else if (f->error() == QFile::OpenError)
            status = QtZipWriter::FileOpenError;
        else if (f->error() == QFile::PermissionsError)
            status = QtZipWriter::FilePermissionsError;
        else
            status = QtZipWriter::FileError;
    }

    d = new QtZipWriterPrivate(f.data(), /*ownDevice=*/true);
    f.take();
    d->status = status;
}
QtZipWriter::QtZipWriter(QIODevice *device)
    : d(new QtZipWriterPrivate(device, /*ownDevice=*/false))
{
    Q_ASSERT(device);
}

QtZipWriter::~QtZipWriter()
{
    close();
    delete d;
}

QIODevice* QtZipWriter::device() const
{
    return d->device;
}

bool QtZipWriter::isWritable() const
{
    return d->device->isWritable();
}

bool QtZipWriter::exists() const
{
    QFile *f = qobject_cast<QFile*> (d->device);
    if (f == 0)
        return true;
    return f->exists();
}
QtZipWriter::Status QtZipWriter::status() const
{
    return d->status;
}

/*!
    \enum QtZipWriter::CompressionPolicy
    \value AlwaysCompress   A file that is added is compressed.
    \value NeverCompress    A file that is added will be stored without changes.
```

```
    \value AutoCompress     A file that is added will be compressed only if that will
give a smaller file.
*/
void QtZipWriter::setCompressionPolicy(CompressionPolicy policy)
{
    d->compressionPolicy = policy;
}
QtZipWriter::CompressionPolicy QtZipWriter::compressionPolicy() const
{
    return d->compressionPolicy;
}
void QtZipWriter::setCreationPermissions(QFile::Permissions permissions)
{
    d->permissions = permissions;
}

/*!
     Returns the currently set creation permissions.
    \sa setCreationPermissions()
    \sa addFile()
*/
QFile::Permissions QtZipWriter::creationPermissions() const
{
    return d->permissions;
}

void QtZipWriter::addFile(const QString &fileName, const QByteArray &data)
{
    d->addEntry(QtZipWriterPrivate::File, QDir::fromNativeSeparators(fileName),
data);
}
void QtZipWriter::addFile(const QString &fileName, QIODevice *device)
{
    Q_ASSERT(device);
    QIODevice::OpenMode mode = device->openMode();
    bool opened = false;
    if ((mode & QIODevice::ReadOnly) == 0) {
        opened = true;
        if (! device->open(QIODevice::ReadOnly)) {
            d->status = FileOpenError;
            return;
        }
    }
    d->addEntry(QtZipWriterPrivate::File, QDir::fromNativeSeparators(fileName),
device->readAll());
    if (opened)
        device->close();
}

/*!
    Create a new directory in the archive with the specified \a dirName and
    the \a permissions;
*/
void QtZipWriter::addDirectory(const QString &dirName)
{
    QString name(QDir::fromNativeSeparators(dirName));
    // separator is mandatory
    if (!name.endsWith(QLatin1Char('/')))
        name.append(QLatin1Char('/'));
    d->addEntry(QtZipWriterPrivate::Directory, name, QByteArray());
}

/*!
```

```
    Create a new symbolic link in the archive with the specified \a dirName
    and the \a permissions;
    A symbolic link contains the destination (relative) path and name.
*/
void QtZipWriter::addSymLink(const QString &fileName, const QString &destination)
{
    d->addEntry(QtZipWriterPrivate::Symlink, QDir::fromNativeSeparators(fileName),
QFile::encodeName(destination));
}

/*!
   Closes the zip file.
*/
void QtZipWriter::close()
{
    if (!(d->device->openMode() & QIODevice::WriteOnly)) {
        return;
    }

    d->device->seek(d->start_of_directory);
    for (int i = 0; i < d->fileHeaders.size(); ++i) {
        const FileHeader &header = d->fileHeaders.at(i);
        d->device->write((const char *)&header.h, sizeof(CentralFileHeader));
        d->device->write(header.file_name);
        d->device->write(header.extra_field);
        d->device->write(header.file_comment);
    }
    int dir_size = d->device->pos() - d->start_of_directory;
    EndOfDirectory eod;
    memset(&eod, 0, sizeof(EndOfDirectory));
    writeUInt(eod.signature, 0x06054b50);
    writeUShort(eod.num_dir_entries_this_disk, d->fileHeaders.size());
    writeUShort(eod.num_dir_entries, d->fileHeaders.size());
    writeUInt(eod.directory_size, dir_size);
    writeUInt(eod.dir_start_offset, d->start_of_directory);
    writeUShort(eod.comment_length, d->comment.length());

    d->device->write((const char *)&eod, sizeof(EndOfDirectory));
    d->device->write(d->comment);
}
```