文本编辑器项目设计说明

本「说明」使用该文本编辑器项目编辑并导出。

1. 需求分析

我们计划完成一个富文本编辑器。结合课程设计要求，我们考虑了如下几个方面的功能和目标；其中亮点使用**蓝色粗体**标出：

a. 文件操作：文件的新建、打开、保存、另存为、**导出为 PDF** 等
b. 文本格式：加粗、斜体、下划线、字号、颜色、字体等
c. 段落格式：左对齐、右对齐、居中对齐等
d. 编辑操作：粘贴（包括**粘贴图片**）、拷贝、剪切、**插入图片**、撤销、重做
e. 查找与替换

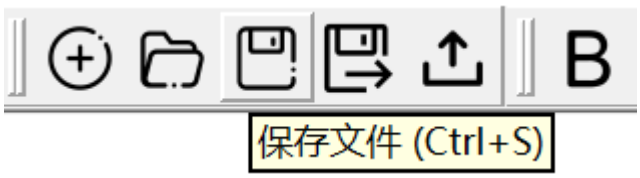
同时，我们考虑了很多关于用户体验的问题。例如上述大多数功能配备了对应的快捷键；在新建、打开文件和关闭窗口前，如果有未保存的修改，询问是否保存等。

2. 总体设计

我们使用 Qt6 作为图形化界面的框架。由于界面比较简单和固定，我们只主要使用了一个 `class MainWindow : public QMainWindow` 作为主界面。

在这个界面中，主要有标题栏（显示当前文件路径）、工具栏和编辑区。界面的大小可以自行调整。

工具栏按照需求分析中的 5 个分类做了分块；每个工具栏可以自由拖动。将鼠标悬浮在工具按钮上，将显示其功能和绑定的快捷键。



文本编辑器项目设计说明

本「说明」使用该文本编辑器项目编辑并导出。

1. 需求分析

我们计划完成一个富文本编辑器。结合课程设计要求，我们考虑了如下几个方面的功能和目标；其中亮点使用**蓝色粗体**标出：

a. 文件操作：文件的新建、打开、保存、另存为、**导出为 PDF** 等
b. 文本格式：加粗、斜体、下划线、字号、颜色、字体等
c. 段落格式：左对齐、右对齐、居中对齐等
d. 编辑操作：粘贴（包括**粘贴图片**）、拷贝、剪切、**插入图片**、撤销、重做
e. 查找与替换

同时，我们考虑了很多关于用户体验的问题。例如上述大多数功能配备了对应的快捷键；在新建、打开文件和关闭窗口前，如果有未保存的修改，询问是否保存等。

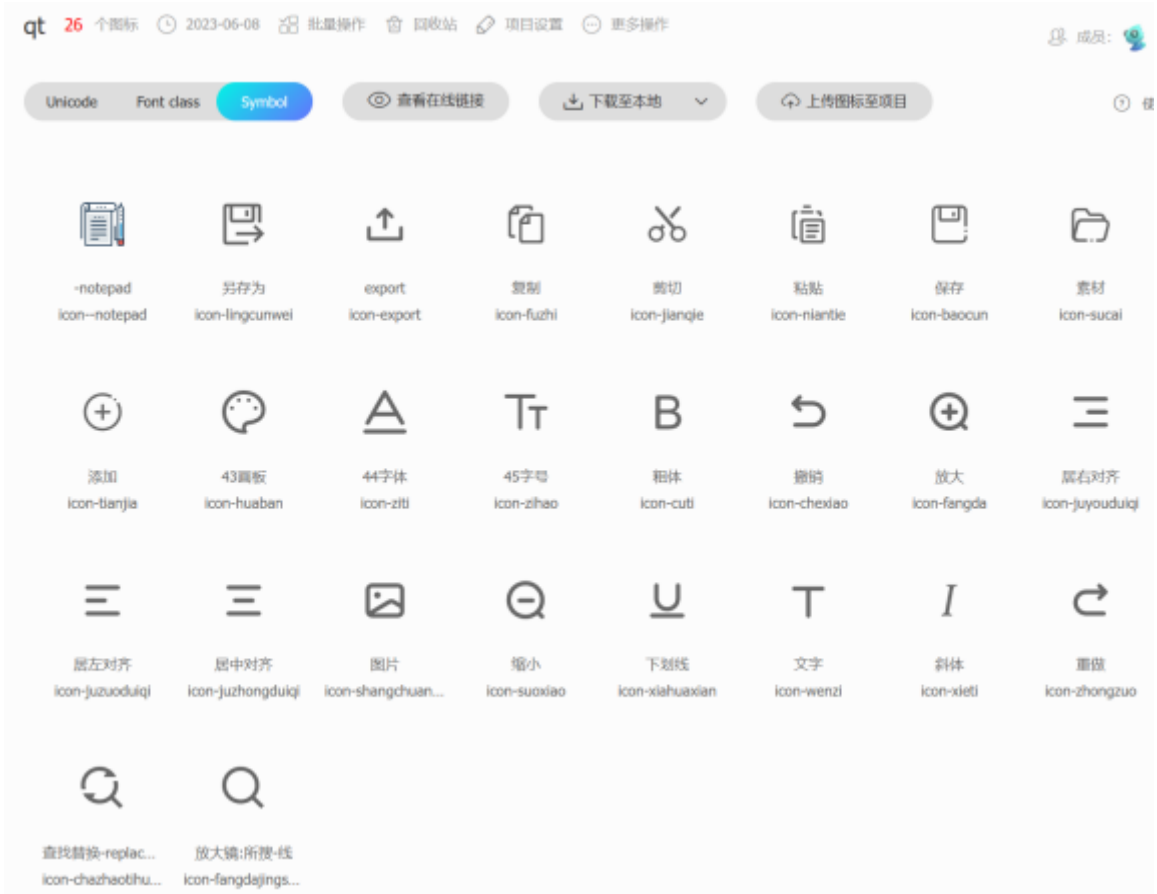
2. 总体设计

我们使用 Qt6 作为图形化界面的框架。由于界面比较简单和固定，我们只主要使用了一个 `class MainWindow : public QMainWindow` 作为主界面。

在这个界面中，主要有标题栏（显示当前文件路径）、工具栏和编辑区。界面的大小可以自行调整；工具栏可以自由拖动。

为了界面美观，我们在 www.iconfont.cn 挑选了要用到的各种功能的按钮图标：

为了界面美观，我们在 www.iconfont.cn 挑选了要用到的各种功能的按钮图标：



3. 系统模块说明

3.1 文本编辑器的界面设计

主界面使用如下类刻画：

```
class MainWindow : public QMainWindow {
public:
    explicit MainWindow(QWidget *parent = nullptr) : QMainWindow(parent) {...}
    void closeEvent(QCloseEvent* event) override;

private:
    void initToolbars();
    // ...

private slots:
    // ...
};
```

在这个类的构造函数中，我们调用函数 `void initToolbars();` 来完成主界面的绘制。这一函数中，我们按照前述讨论，创建了 5 个工具栏，并添加了对应的按钮 `QToolBar`，绑定了对应的槽函数。

3.2 存储相关处理

[TODO]

3.3 文本和段落格式相关处理

[TODO]

3.4 编辑相关处理

[TODO]

3.5 查找替换相关处理

[TODO]

4. 系统设计难点及其解决

导出 PDF 并不是 QTextDocument 自带的功能。我们查阅资料后，本来选择了 QPdfWriter，但是以这种方式只能打印在单页 PDF 上。因此后来我们改用了 QPrinter。

加入「导出PDF」功能过程中遇到的一个问题是，程序通过了编译，但运行出现「Process finished with exit code -1073741515 (0xC0000135)」的提示。经查阅资料，发现是 dll 找不到。经过排查后，发现 CMake 并没有把 Qt6PrintSupport.dll 正确复制到工程目录下。我们使用手动复制的方法暂时解决了这一问题。

另一个问题是，由于 QPrinter 并不支持打印时进行缩放，因此图片经常会超宽。为了避免这种问题，我们在插入图片时新增了逻辑：如果图片宽度大于导出 PDF 的宽度，则对其进行等比例缩放，使其宽度变为适合打印的宽度。

同时，粘贴剪切板中的图片也是一个困难的问题；网络上关于此的资料相对比较少。QTextEdit 并不支持这一功能。我们研究了 `QGuiApplication` 的相关接口，结合 mimeData 完成了剪切板中图片的识别和加载，然后将图片保存在本地后插入到文档中。为了支持快捷键，我们不再使用 QTextEdit，而是创建了一个继承自它的新类 MyTextEdit 从而屏蔽 QTextEdit 对粘贴快捷键 Ctrl+V 的绑定：

```
class MyTextEdit : public QTextEdit {
public:
    explicit MyTextEdit(QWidget *parent = nullptr) : QTextEdit(parent) {}
    void customPaste();
protected:
    void keyPressEvent(QKeyEvent *event) override {
        if (event->modifiers() == Qt::ControlModifier && event->key() ==
Qt::Key_V) {
            // 处理 Ctrl + V 快捷键
            customPaste();
            return;
        }

        // 其他情况下，继续处理默认的键盘事件
        QTextEdit::keyPressEvent(event);
    }
};
```

5. 总结

总结而言，本工作实现了题目要求的所有基础目标和关键目标，并额外提供了导出 PDF、图片等功能。与此同时，本工作在使用 Qt 提供的 QTextDocument 和 QTextEdit 的同时，为了实现额外功能，对其做了扩展和丰富。

在此过程中，我们在设计上做了充分的调研、准备和探索，对 Qt 有了比较

深入的学习和讨论，对面向对象编程有了更加深刻的认识。

程序使用说明

[TODO]

系统开发日志

由于时间或条件限制，以下问题暂时没有解决：

1. 窗口缩放功能

我们试图增加 Microsoft Word 那样的放大缩小功能；但是 QTextEdit 自带的 zoomIn 和 zoomOut 函数的实现方式实际上是更改字号；这对于我们的 HTML 富文本来说是没有用的。

我们探索使用 CSS 来解决这个问题，并编写了这样的代码：

```
void MainWindow::zoom(qreal delta) {
    static qreal zoomFactor = 1.0;
    zoomFactor += delta;
    QString css = QString("body { transform: scale(%1);
}").arg(zoomFactor);
    textEdit->document()->setDefaultStyleSheet(css);
}
```

不过仍然没有成功。查阅资料后，我们在 https://doc.qt.io/qt-6/richtext-html-subset.html#css-properties 发现，目前 QTextEdit 所支持的 CSS 子集中并没有能完成这一工作的方法。

进一步查阅资料后，我们发现网络上对此问题的实现大多仍然是调整字号，例如 https://stackoverflow.com/questions/8016530/no-effect-from-zoomin-in-qtextedit-after-font-size-change 。而这与我们其他部分的设计思路不符。最终，我们放弃了这一设计。

2. 图片质量问题

根据我们的设计，我们会在导入图片时对其进行缩放。但是，Qt 的相关功能会使得图片质量受损。我们暂时没有找到这个问题的解决方案。

3. 含图片的文本粘贴问题

本工作能够实现程序内部的含图片文本粘贴，但对于其他来源的含图片文本，我们并不能完成处理。这是因为，我们处理图片粘贴是手动使用 `QGuiApplication` 相关接口实现的，而并不是使用 QTextEdit 原生支持的手段。如果我们希望支持其他来源的含图片文本，则需要对文本进行遍历和解析；这是比较难以实现的。因此，我们没有实现这一功能。