基于 Qt 的记事本项目

一、需求文档

1. 核心功能

文本编辑：支持多行文本输入、复制/粘贴/剪切/全选

文件操作：新建/打开/保存/另存为（支持.txt 格式）

格式设置：字体类型、字号、颜色选择

查找替换：支持区分大小写和全词匹配

状态显示：当前光标位置（行号/列号）、文件编码格式

基础交互：撤销/重做、关于对话框

2. 扩展功能（可选）

多标签页编辑

语法高亮（支持简单编程语言）

自动保存与恢复

快捷键自定义

二、技术架构

1. 架构模式

```
A[MainWindow] --> B[UI 模块]
    A --> C[业务逻辑模块]
    A --> D[文件操作模块]
    C --> E[编辑功能]
    C --> F[查找替换]
    D --> G[编码转换]
    D --> H[异常处理]
```

2. 技术栈

Qt 5.12+（LGPL 协议）
C++11 标准
核心组件：

```
QMainWindow（主窗口）
QTextEdit（文本编辑区）
QMenu/QToolBar（界面元素）
QSettings（配置存储）
QSyntaxHighlighter（语法高亮，可选）...
```

三、实现步骤

1. 环境搭建

```
#  使用 Qt Creator 创建新项目
File -> New Project -> Qt Widgets Application
#  选择 QMainWindow 作为基类
```

2. 界面开发

```
// mainwindow.h
class MainWindow : public QMainWindow {
    Q_OBJECT
public:
    // ...构造函数等声明...
private:
    QTextEdit *textEdit;
    QLabel *statusLabel;

    void createMenus();
    void createToolBars();
    void createStatusBar();
};
```

3. 核心功能实现

文件操作示例：

```
// 保存文件实现
void MainWindow::saveFile() {
    QFile file(currentFile);
    if (!file.open(QIODevice::WriteOnly | QIODevice::Text)) {
        QMessageBox::warning(this, "Error", "Cannot save file");
```

```
        return;
    }

    QTextStream out(&file);
    out.setCodec("UTF-8");   // 处理编码
    out << textEdit->toPlainText();
    file.close();

    setWindowModified(false);
}
```

查找替换实现：

```
void MainWindow::showFindDialog() {
    QDialog dialog(this);
    QLineEdit *findLineEdit = new QLineEdit(&dialog);
    QCheckBox *caseCheckBox = new QCheckBox("Match case", &dialog);

    connect(findLineEdit, &QLineEdit::returnPressed, [&](){
        findText(findLineEdit->text(),
                 caseCheckBox->isChecked());
    });

    dialog.exec();
}
```

4. 信号槽连接

```
// 连接撤销操作
connect(undoAction, &QAction::triggered,
        textEdit, &QTextEdit::undo);

// 文本修改状态跟踪
connect(textEdit->document(), &QTextDocument::modificationChanged,
        this, &MainWindow::documentModified);
```

四、关键实现细节

1. 编码处理

```
// 自动检测文件编码
QTextCodec *codec = QTextCodec::codecForUtfText(file.peek(1024),
            QTextCodec::codecForName("GBK"));
```

2. 状态栏更新

```
void MainWindow::updateCursorPos() {
    QTextCursor cursor = textEdit->textCursor();
    statusLabel->setText(QString("Ln %1, Col %2")
                         .arg(cursor.blockNumber()+1)
                         .arg(cursor.columnNumber()+1));
}
```

五、测试方案

1. 单元测试用例

```
// 测试文件保存功能
void TestNotepad::testSave() {
    MainWindow w;
    w.textEdit->setPlainText("Test content");
    w.saveAs("test.txt");

    QFile file("test.txt");
    QVERIFY(file.exists());
    QCOMPARE(file.readAll(), QByteArray("Test content"));
}
```

2. 界面自动化测试

使用 Qt Test 模块实现：

```
QTest::keyClicks(textEdit, "Hello World");
QTest::mouseClick(saveButton, Qt::LeftButton);
```

六、项目扩展建议

插件系统：通过动态库实现功能扩展

版本控制：集成 Git 基础操作

AI 集成：接入开源 NLP 模型实现智能补全

七、学习资源推荐

Qt 官方文档：https://doc.qt.io

项目代码模板：GitHub 搜索"qt-notepad-example"

调试技巧：使用 qDebug()输出日志

建议开发周期：1 周（基础功能）+ 1 周（扩展功能）