PROJECT REPORT

A Digital Notepad With Superpowers

Submitted by: *Nakshatra Mudgil (23CSU207)*

📝*Problem Statement: A Digital Notepad with Superpowers*

Build a simple yet powerful text-editor application that allows users to create, open, edit, and save text files—similar to a basic Notepad.
The system should provide a clean interface, essential editing features, and reliable file handling for everyday writing tasks.

The goal is to deliver a lightweight tool that helps users quickly jot down notes, draft ideas, or edit plain text without unnecessary complexity

1. Introduction

This project is a **GUI-based digital notepad application** designed to provide students and users with a simple yet powerful text-editing tool. Inspired by Windows Notepad, this application allows users to create, edit, and manage text files with ease.

The notepad goes beyond basic functionality by offering **supercharged features** such as keyboard shortcuts, enhanced editing tools, and an intuitive user interface. The application is built using **C++ and the Qt5 framework**, ensuring fast performance and cross-platform compatibility.

2. Project Objective

The main objective of this project is to:

- Build a **lightweight and efficient text editor**.
- Implement core features found in modern text editors.
- Provide a clean GUI for editing text files.
- Develop a real-world, practical application using C++ and Qt.
- Practice GUI development, file handling, and event-driven programming.

3. Technologies Used

Component	Technology
Programming Language	**C++**
GUI Framework	**Qt 5 (QtWidgets, QtGui, QtCore)**
Development Editor	**Vim**

Component	Technology
Operating System	**Linux (Ubuntu/Debian/Arch-based)**
Build Tool	**qmake + g++**

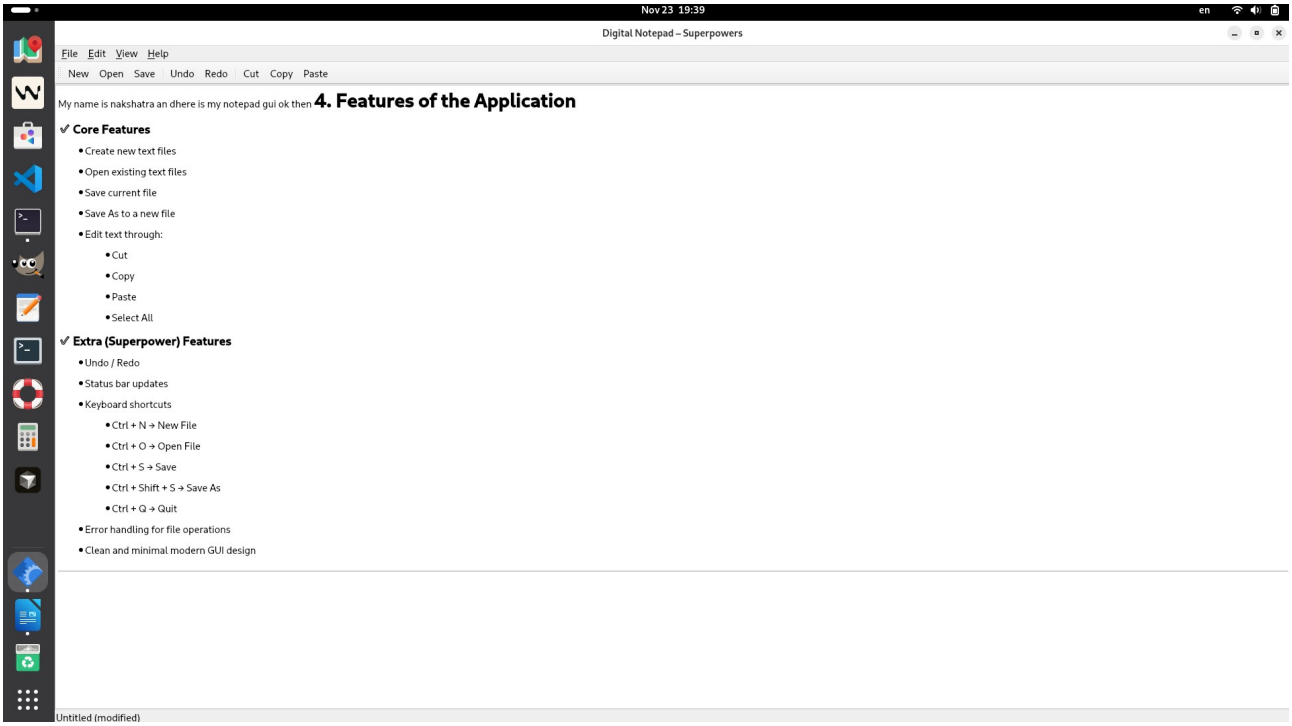
4. Features of the Application

✅ Core Features

- Create new text files
- Open existing text files
- Save current file
- Save As to a new file
- Edit text through:
    - Cut
    - Copy
    - Paste
    - Select All

✅ Extra (Superpower) Features

- Undo / Redo
- Status bar updates
- Keyboard shortcuts
    - Ctrl + N → New File
    - Ctrl + O → Open File
    - Ctrl + S → Save
    - Ctrl + Shift + S → Save As
    - Ctrl + Q → Quit
- Error handling for file operations
- Clean and minimal modern GUI design



code in vim editor

```
#!/bin/bash
set -e

echo "🌟 Setting up Digital Notepad with Superpowers..."

PROJECT_DIR="super-notepad"
mkdir -p "$PROJECT_DIR"
cd "$PROJECT_DIR"

# --- notepadgui.h ---
cat > notepadgui.h <<'EOF'
#ifndef NOTEPADGUI_H
#define NOTEPADGUI_H

#include <QMainWindow>

class QTextEdit;
class QLabel;

class NotepadGUI : public QMainWindow {
    Q_OBJECT
public:
    explicit NotepadGUI(QWidget *parent = nullptr);
    ~NotepadGUI();

private slots:
    // File actions
    void newFile();
    void openFile();
    bool saveFile();
    bool saveAsFile();
    void maybeSaveAndNew();

    // Edit actions
    void cut();
    void copy();
    void paste();
    void selectAll();

    // Find / Replace
    void findText();
    void replaceText();

    // Other utilities
    void updateStatus();
    void toggleDarkMode();
    void showAbout();

private:
    void createActions();
-- INSERT --                                                    45,1          Top
```

```
private:
    void createActions();
    void createMenus();
    void createToolBar();
    void createStatusBar();
    bool maybeSave();
    bool writeToFile(const QString &fileName);
    bool readFromFile(const QString &fileName);
    void setCurrentFile(const QString &fileName);
    QString strippedName(const QString &fullFileName);

    QTextEdit *textEdit;
    QString currentFile;
    QLabel *statusLabel;

    // Actions
    QAction *actionNew;
    QAction *actionOpen;
    QAction *actionSave;
    QAction *actionSaveAs;
    QAction *actionExit;

    QAction *actionCut;
    QAction *actionCopy;
    QAction *actionPaste;
    QAction *actionSelectAll;
    QAction *actionUndo;
    QAction *actionRedo;

    QAction *actionFind;
    QAction *actionReplace;
    QAction *actionDarkMode;
    QAction *actionAbout;
};

#endif // NOTEPADGUI_H
EOF

# --- notepadgui.cpp ---
cat > notepadgui.cpp <<'EOF'
#include "notepadgui.h"
#include <QTextEdit>
#include <QMenuBar>
#include <QToolBar>
#include <QFileDialog>
#include <QMessageBox>
#include <QStatusBar>
#include <QLabel>
#include <QAction>
-- INSERT --                                                                                                    92,1          11%
```

```
#endif // NOTEPADGUI_H
EOF

# --- notepadgui.cpp ---
cat > notepadgui.cpp <<'EOF'
#include "notepadgui.h"
#include <QTextEdit>
#include <QMenuBar>
#include <QToolBar>
#include <QFileDialog>
#include <QMessageBox>
#include <QStatusBar>
#include <QLabel>
#include <QAction>
#include <QApplication>
#include <QFile>
#include <QTextStream>
#include <QKeySequence>
#include <QInputDialog>

NotepadGUI::NotepadGUI(QWidget *parent)
    : QMainWindow(parent), textEdit(new QTextEdit(this)), statusLabel(new QLabel(this))
{
    setCentralWidget(textEdit);
    createActions();
    createMenus();
    createToolBar();
    createStatusBar();

    setCurrentFile(QString());
    resize(900, 600);
    setWindowTitle("Digital Notepad — Superpowers");
    connect(textEdit->document(), &QTextDocument::modificationChanged, this, &NotepadGUI::updateStatus);
}

NotepadGUI::~NotepadGUI() {}

void NotepadGUI::createActions()
{
    // File
    actionNew = new QAction("&New", this);
    actionNew->setShortcut(QKeySequence::New);
    connect(actionNew, &QAction::triggered, this, &NotepadGUI::newFile);

    actionOpen = new QAction("&Open...", this);
    actionOpen->setShortcut(QKeySequence::Open);
    connect(actionOpen, &QAction::triggered, this, &NotepadGUI::openFile);

    actionSave = new QAction("&Save", this);
-- INSERT --                                                                                                   127,1          20%
```

```
NotepadGUI::~NotepadGUI() {}

void NotepadGUI::createActions()
{
    // File
    actionNew = new QAction("&New", this);
    actionNew->setShortcut(QKeySequence::New);
    connect(actionNew, &QAction::triggered, this, &NotepadGUI::newFile);

    actionOpen = new QAction("&Open...", this);
    actionOpen->setShortcut(QKeySequence::Open);
    connect(actionOpen, &QAction::triggered, this, &NotepadGUI::openFile);

    actionSave = new QAction("&Save", this);
    actionSave->setShortcut(QKeySequence::Save);
    connect(actionSave, &QAction::triggered, this, &NotepadGUI::saveFile);

    actionSaveAs = new QAction("Save &As...", this);
    actionSaveAs->setShortcut(QKeySequence(Qt::CTRL + Qt::SHIFT + Qt::Key_S));
    connect(actionSaveAs, &QAction::triggered, this, &NotepadGUI::saveAsFile);

    actionExit = new QAction("E&xit", this);
    actionExit->setShortcut(QKeySequence::Quit);
    connect(actionExit, &QAction::triggered, qApp, &QApplication::closeAllWindows);

    // Edit
    actionUndo = new QAction("&Undo", this);
    actionUndo->setShortcut(QKeySequence::Undo);
    connect(actionUndo, &QAction::triggered, textEdit, &QTextEdit::undo);

    actionRedo = new QAction("&Redo", this);
    actionRedo->setShortcut(QKeySequence::Redo);
    connect(actionRedo, &QAction::triggered, textEdit, &QTextEdit::redo);

    actionCut = new QAction("Cu&t", this);
    actionCut->setShortcut(QKeySequence::Cut);
    connect(actionCut, &QAction::triggered, this, &NotepadGUI::cut);

    actionCopy = new QAction("&Copy", this);
    actionCopy->setShortcut(QKeySequence::Copy);
    connect(actionCopy, &QAction::triggered, this, &NotepadGUI::copy);

    actionPaste = new QAction("&Paste", this);
    actionPaste->setShortcut(QKeySequence::Paste);
    connect(actionPaste, &QAction::triggered, this, &NotepadGUI::paste);

    actionSelectAll = new QAction("Select &All", this);
    actionSelectAll->setShortcut(QKeySequence::SelectAll);
    connect(actionSelectAll, &QAction::triggered, this, &NotepadGUI::selectAll);
-- INSERT --                                                                                                    162,1         28%
```

```
}

void NotepadGUI::createMenus()
{
    QMenu *fileMenu = menuBar()->addMenu("&File");
    fileMenu->addAction(actionNew);
    fileMenu->addAction(actionOpen);
    fileMenu->addAction(actionSave);
    fileMenu->addAction(actionSaveAs);
    fileMenu->addSeparator();
    fileMenu->addAction(actionExit);

    QMenu *editMenu = menuBar()->addMenu("&Edit");
    editMenu->addAction(actionUndo);
    editMenu->addAction(actionRedo);
    editMenu->addSeparator();
    editMenu->addAction(actionCut);
    editMenu->addAction(actionCopy);
    editMenu->addAction(actionPaste);
    editMenu->addAction(actionSelectAll);
    editMenu->addSeparator();
    editMenu->addAction(actionFind);
    editMenu->addAction(actionReplace);

    QMenu *viewMenu = menuBar()->addMenu("&View");
    viewMenu->addAction(actionDarkMode);

    QMenu *helpMenu = menuBar()->addMenu("&Help");
    helpMenu->addAction(actionAbout);
}

void NotepadGUI::createToolBar()
{
    QToolBar *tb = addToolBar("Main");
    tb->addAction(actionNew);
    tb->addAction(actionOpen);
    tb->addAction(actionSave);
    tb->addSeparator();
    tb->addAction(actionUndo);
    tb->addAction(actionRedo);
    tb->addSeparator();
    tb->addAction(actionCut);
    tb->addAction(actionCopy);
    tb->addAction(actionPaste);
}

void NotepadGUI::createStatusBar()
{
    statusBar()->addWidget(statusLabel, 1);
    updateStatus();
-- INSERT --                                                                                                    227,1         44%
```

```
}

void NotepadGUI::createStatusBar()
{
    statusBar()->addWidget(statusLabel, 1);
    updateStatus();
}

void NotepadGUI::updateStatus()
{
    QString status;
    if (currentFile.isEmpty()) status = "Untitled";
    else status = strippedName(currentFile);

    if (textEdit->document()->isModified()) status += " (modified)";

    statusLabel->setText(status);
}

void NotepadGUI::newFile()
{
    if (!maybeSave()) return;
    textEdit->clear();
    setCurrentFile(QString());
}

void NotepadGUI::openFile()
{
    if (!maybeSave()) return;
    QString fileName = QFileDialog::getOpenFileName(this, "Open File");
    if (!fileName.isEmpty()) {
        if (readFromFile(fileName))
            setCurrentFile(fileName);
    }
}

bool NotepadGUI::saveFile()
{
    if (currentFile.isEmpty()) return saveAsFile();
    return writeToFile(currentFile);
}

bool NotepadGUI::saveAsFile()
{
    QString fileName = QFileDialog::getSaveFileName(this, "Save File As");
    if (fileName.isEmpty()) return false;
    if (writeToFile(fileName)) {
        setCurrentFile(fileName);
        return true;
    }
-- INSERT --                                                                                                    271,1         55%
```

```
bool NotepadGUI::saveAsFile()
{
    QString fileName = QFileDialog::getSaveFileName(this, "Save File As");
    if (fileName.isEmpty()) return false;
    if (writeToFile(fileName)) {
        setCurrentFile(fileName);
        return true;
    }
    return false;
}

void NotepadGUI::maybeSaveAndNew()
{
    if (maybeSave()) {
        newFile();
    }
}

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

    QMessageBox::StandardButton ret =
        QMessageBox::warning(this, "Digital Notepad",
                             "The document has been modified.\nDo you want to save your changes?",
                             QMessageBox::Save | QMessageBox::Discard | QMessageBox::Cancel);
    if (ret == QMessageBox::Save) return saveFile();
    if (ret == QMessageBox::Cancel) return false;
    return true; // Discard
}

bool NotepadGUI::writeToFile(const QString &fileName)
{
    QFile file(fileName);
    if (!file.open(QFile::WriteOnly | QFile::Text)) {
        QMessageBox::warning(this, "Digital Notepad",
                             QString("Cannot write file %1:\n%2.").arg(fileName, file.errorString()));
        return false;
    }

    QTextStream out(&file);
    out << textEdit->toPlainText();
    file.close();
    textEdit->document()->setModified(false);
    updateStatus();
    return true;
}

bool NotepadGUI::readFromFile(const QString &fileName)
-- INSERT --                                                                                                    312,1         65%
```

```
bool NotepadGUI::readFromFile(const QString &fileName)
{
    QFile file(fileName);
    if (!file.open(QFile::ReadOnly | QFile::Text)) {
        QMessageBox::warning(this, "Digital Notepad",
                             QString("Cannot read file %1:\n%2.").arg(fileName, file.errorString()));
        return false;
    }

    QTextStream in(&file);
    textEdit->setPlainText(in.readAll());
    textEdit->document()->setModified(false);
    file.close();
    updateStatus();
    return true;
}

void NotepadGUI::setCurrentFile(const QString &fileName)
{
    currentFile = fileName;
    textEdit->document()->setModified(false);
    updateStatus();
    setWindowTitle(QString("%1 - Digital Notepad").arg(currentFile.isEmpty() ? "Untitled" : strippedName(currentFile)));
}

QString NotepadGUI::strippedName(const QString &fullFileName)
{
    return QFileInfo(fullFileName).fileName();
}

void NotepadGUI::cut() { textEdit->cut(); }
void NotepadGUI::copy() { textEdit->copy(); }
void NotepadGUI::paste() { textEdit->paste(); }
void NotepadGUI::selectAll() { textEdit->selectAll(); }

void NotepadGUI::findText()
{
    bool ok;
    QString text = QInputDialog::getText(this, "Find", "Find:", QLineEdit::Normal, "", &ok);
    if (!ok || text.isEmpty()) return;

    // simple find: move cursor to first match
    QTextDocument *doc = textEdit->document();
    QTextCursor cursor = doc->find(text, 0);
    if (!cursor.isNull()) {
        textEdit->setTextCursor(cursor);
    } else {
        QMessageBox::information(this, "Find", "No matches found.");
    }
}
-- INSERT --                                                                                                    361,1         77%
```

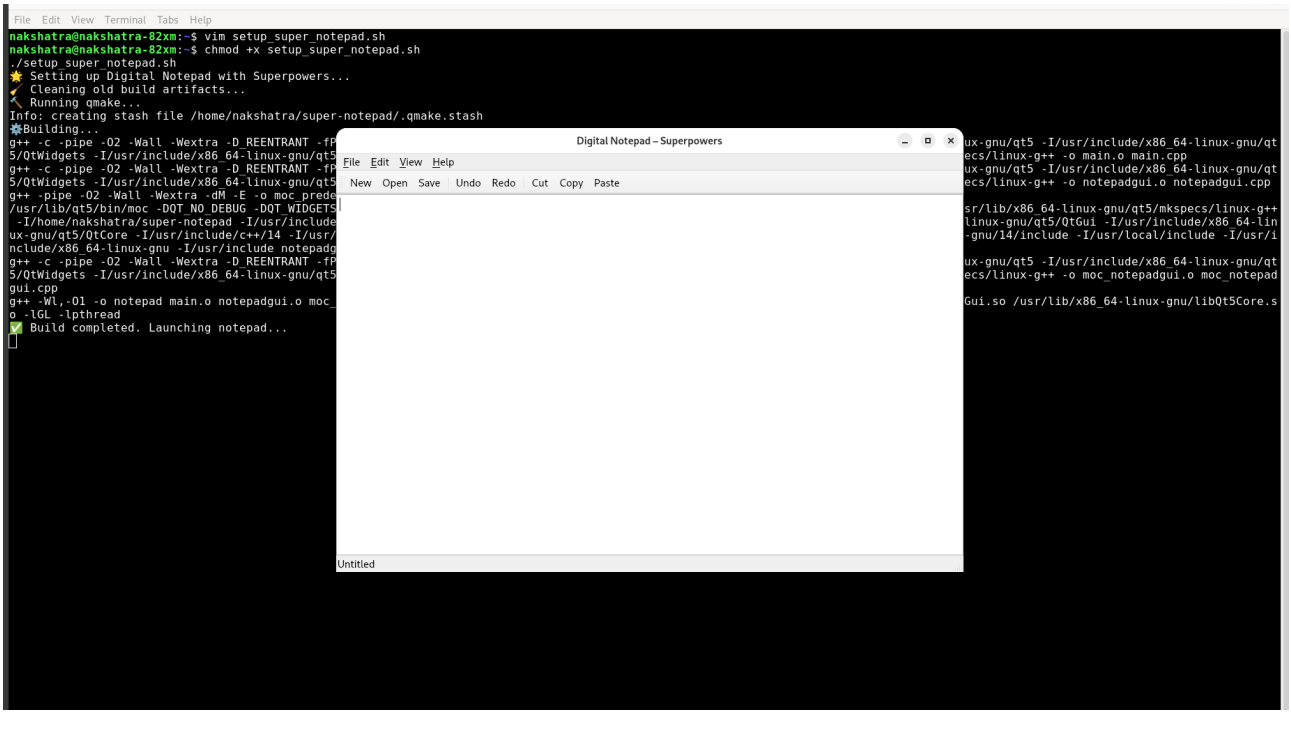
```
void NotepadGUI::replaceText()
{
    bool ok;
    QString findStr = QInputDialog::getText(this, "Replace", "Find:", QLineEdit::Normal, "", &ok);
    if (!ok || findStr.isEmpty()) return;
    QString replaceStr = QInputDialog::getText(this, "Replace", "Replace with:", QLineEdit::Normal, "", &ok);
    if (!ok) return;

    // Replace all occurrences in entire document (simple)
    QString content = textEdit->toPlainText();
    int occurrences = 0;
    int pos = 0;
    while ((pos = content.indexOf(findStr, pos, Qt::CaseSensitive)) != -1) {
        content.replace(pos, findStr.length(), replaceStr);
        pos += replaceStr.length();
        occurrences++;
    }

    if (occurrences == 0) {
        QMessageBox::information(this, "Replace", "No matches found.");
    } else {
        textEdit->setPlainText(content);
        QMessageBox::information(this, "Replace", QString("Replaced %1 occurrence(s).").arg(occurrences));
    }
}

void NotepadGUI::toggleDarkMode()
{
    if (actionDarkMode->isChecked()) {
        QString style = "QWidget{ background: #121212; color: #e0e0e0 }"
                        "QTextEdit { background: #1e1e1e; color: #e0e0e0 }"
                        "QMenuBar { background: #151515; color: #e0e0e0 }"
                        "QMenu { background: #151515; color: #e0e0e0 }"
                        "QToolBar { background: #151515; color: #e0e0e0 }";
        qApp->setStyleSheet(style);
    } else {
        qApp->setStyleSheet("");
    }
}

void NotepadGUI::showAbout()
{
    QMessageBox::about(this, "About Digital Notepad",
                       "Digital Notepad with Superpowers\n"
                       "Features: New/Open/Save/Save As, Undo/Redo, Cut/Copy/Paste, Find/Replace, Dark Mode.\n"
                       "Built with Qt.");
}
EOF
-- INSERT --                                                                                                    411,1         89%
```

```
    QMessageBox::about(this, "About Digital Notepad",
                       "Digital Notepad with Superpowers\n"
                       "Features: New/Open/Save/Save As, Undo/Redo, Cut/Copy/Paste, Find/Replace, Dark Mode.\n"
                       "Built with Qt.");
}
EOF

# --- main.cpp ---
cat > main.cpp <<'EOF'
#include <QApplication>
#include "notepadgui.h"

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

# --- notepad.pro ---
cat > notepad.pro <<'EOF'
QT += widgets

TARGET = notepad

SOURCES += main.cpp \
           notepadgui.cpp

HEADERS += notepadgui.h
EOF

echo "🧹 Cleaning old build artifacts..."
rm -f Makefile .qmake.stash moc_* *.o notepad

echo "🔨 Running qmake..."
qmake

echo "⚙️Building..."
make -j$(nproc)

if [ -f notepad ]; then
    echo "✅ Build completed. Launching notepad..."
    ./notepad
else
    echo "❌ Build failed."
    exit 1
fi
-- INSERT --                                                                    459,1         Bot
```

Make it executable and run:

```
chmod +x setup_super_notepad.sh
./setup_super_notepad.sh
```

final output of gui text editor:-



Additional Data Structures Used Indirectly

Feature	Data Structure Used	Why
Text editor	**Tree of text fragments**	Efficient editing
File paths	`QString`	Unicode-safe string class
File operations	`QFile`	File abstraction
Reading/writing text	`QTextStream`	Stream buffer (queue-like reading)
GUI components	QObject hierarchy	Tree structure

Tree-Based Text Fragment Structure (Main Data Structure)

Used internally by `QTextDocument` to store and format text.

✔️ Efficient editing
✔️ Fast insert/delete
✔️ Used by most professional editors

2. QString (Dynamic Array / Unicode String)

Qt's `QString` behaves like a **dynamic array of Unicode characters**.

Used for:

- File path
- Text content
- Title updates
- Status bar messages

3. QVector (Dynamic Array)

Used internally by Qt for storing UI components, actions, menu items, etc.

4. QList (Linked List / Dynamic List)

Qt uses this for managing objects such as:

- Actions in menu bar
- Child widgets
- Undo/redo stacks

5. QStack (Stack Data Structure)

Used by Qt's Undo/Redo system.

✔️ LIFO (Last In First Out)
✔️ Undo pushes an operation
✔️ Redo pops the last operation

6. QFile + QTextStream (Stream Buffer / Queue-like)

Used for reading and writing files.

✔️ Buffered reading
✔️ Sequential processing
✔️ Easy line-by-line read/write

“This project uses multiple data structures including Qt’s internal tree-based text fragment model (used by QTextDocument), dynamic arrays (QString, QVector), lists (QList), stack (QStack for undo/redo), and stream buffers (QTextStream for file handling). These data structures together provide efficient text editing, file processing, and GUI management.”