项目名称：MiniIDE – C语言编辑器

文档编号：2.0

MiniIDE – C语言编辑器

【设计文档】

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 总页数 |  | 正文 |  | 附录 |  | 生效日期 |  |
| 编制 |  | | | 批准 |  | | |

目录

[1 整体架构设计 5](#_Toc19126355)

[2 界面设计方案 6](#_Toc19126356)

[2.1 界面整体布局设计方案 6](#_Toc19126357)

[2.1.1 界面整体布局设计图 6](#_Toc19126358)

[2.1.2 界面整体布局预览 6](#_Toc19126359)

[2.2 标题栏设计方案 7](#_Toc19126360)

[2.3 菜单栏设计方案 7](#_Toc19126361)

[2.3.1 菜单栏整体设计方案 7](#_Toc19126362)

[2.3.2 文件区域设计方案 7](#_Toc19126363)

[2.3.3 编辑区域设计方案 8](#_Toc19126364)

[2.3.4 编译运行区域设计方案 8](#_Toc19126365)

[2.3.5 帮助区域设计方案 9](#_Toc19126366)

[2.3.6 工具栏设计方案 10](#_Toc19126367)

[2.4 代码编辑区设计方案 11](#_Toc19126368)

[2.5 文件资源管理区设计方案 11](#_Toc19126369)

[2.6 状态栏设计方案 12](#_Toc19126370)

[2.7 控制台设计方案 12](#_Toc19126371)

[3 数据结构设计（UML设计） 13](#_Toc19126372)

[3.1 各功能设计 13](#_Toc19126373)

[3.1.1 编辑器 13](#_Toc19126374)

[3.1.2 编译功能 13](#_Toc19126375)

[3.1.3 调试功能 13](#_Toc19126376)

[3.1.4 窗口功能 13](#_Toc19126377)

[3.1.5 按钮监听功能 13](#_Toc19126378)

[3.2 UML图 14](#_Toc19126379)

[4 系统结构图 15](#_Toc19126380)

[5 程序功能表 16](#_Toc19126381)

[6 关键算法设计 17](#_Toc19126382)

[6.1 文本编辑模块 17](#_Toc19126383)

[6.1.1 复制 17](#_Toc19126384)

[6.1.2 粘贴 17](#_Toc19126385)

[6.1.3 查找 17](#_Toc19126386)

[6.1.4 替换 18](#_Toc19126387)

[6.1.5 撤销 20](#_Toc19126388)

[6.1.6 重做 20](#_Toc19126389)

[6.1.7 显示当前行数和列号 20](#_Toc19126390)

[6.1.8 调整字体大小 21](#_Toc19126391)

[6.2 代码编辑模块 23](#_Toc19126392)

[6.2.1 关键字识别高亮 23](#_Toc19126393)

[6.2.2 关键字联想提示、补全 23](#_Toc19126394)

[6.2.3 括号自动匹配高亮 24](#_Toc19126395)

[6.2.4 符号自动补全 24](#_Toc19126396)

[6.2.5 变量重命名 25](#_Toc19126397)

[6.2.6 自动缩进 25](#_Toc19126398)

[6.2.7 多行注释 27](#_Toc19126399)

[6.2.8 注释隐藏 / 显示 28](#_Toc19126400)

[6.2.9 代码块折叠 29](#_Toc19126401)

[6.2.10 对已编辑的函数进行高亮显示 29](#_Toc19126402)

[6.2.11 代码跳转 30](#_Toc19126403)

[6.2.12 行格式排版 31](#_Toc19126404)

[6.2.13 整体格式排版 32](#_Toc19126405)

[6.3 编译运行模块 34](#_Toc19126406)

[6.3.1 通过gcc对当前文件编译 34](#_Toc19126407)

[6.3.2 通过gcc对当前文件运行 35](#_Toc19126408)

[6.3.3 通过gcc实现多文件编译 35](#_Toc19126409)

[6.3.4 通过gcc实现项目运行 37](#_Toc19126410)

[6.3.5 在控制台显示编译结果 37](#_Toc19126411)

[6.3.6 通过gdb对当前文件调试 39](#_Toc19126412)

[6.4 文件管理模块 41](#_Toc19126413)

[6.4.1 新建文件 41](#_Toc19126414)

[6.4.2 打开文件 42](#_Toc19126415)

[6.4.3 打开文件夹 43](#_Toc19126416)

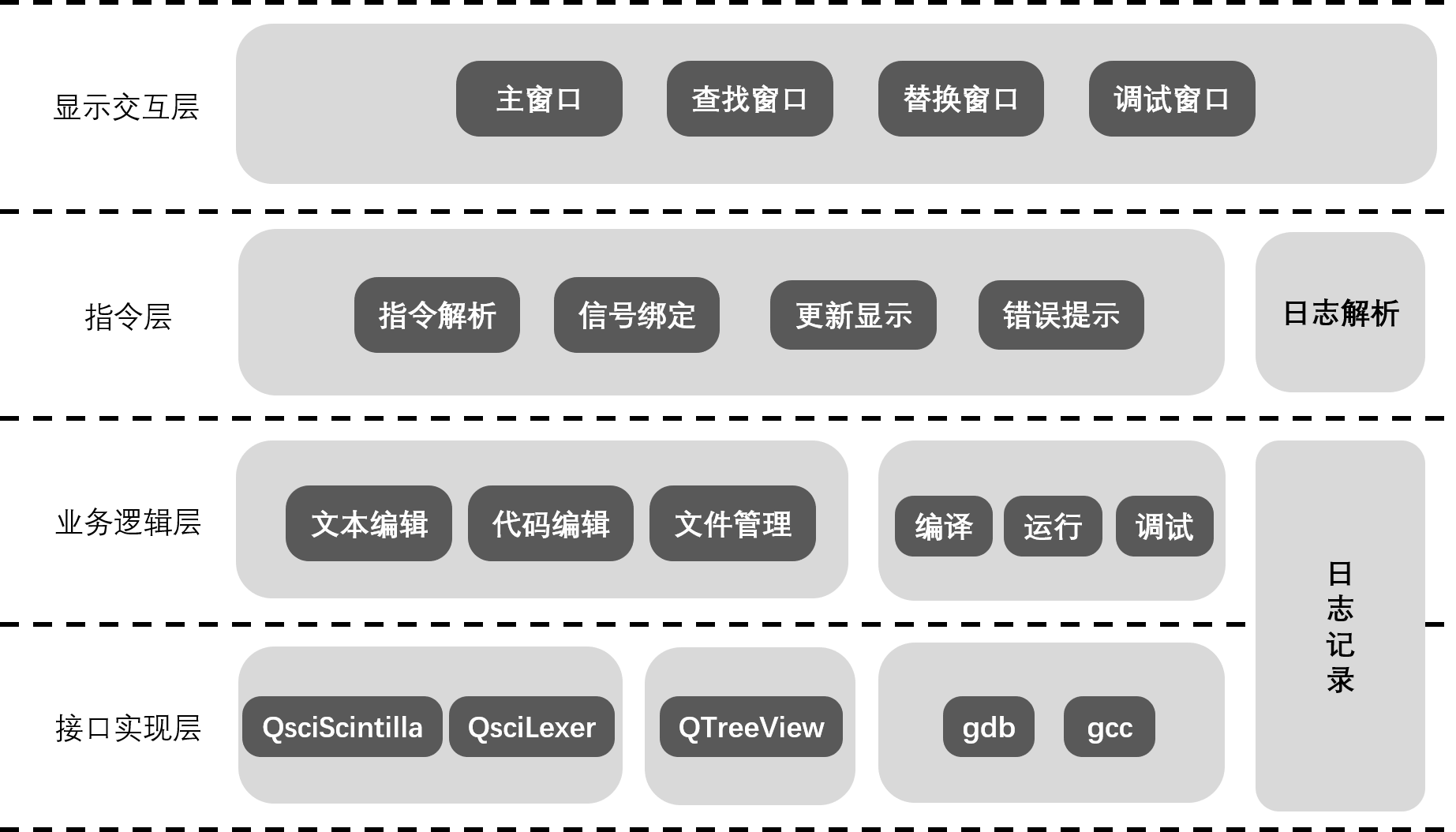
[6.4.4 保存文件 43](#_Toc19126417)

[6.4.5 另存为文件 45](#_Toc19126418)

[6.4.6 多文件编辑 45](#_Toc19126419)

[6.4.7 树形文件资源管理 46](#_Toc19126420)

# 整体架构设计



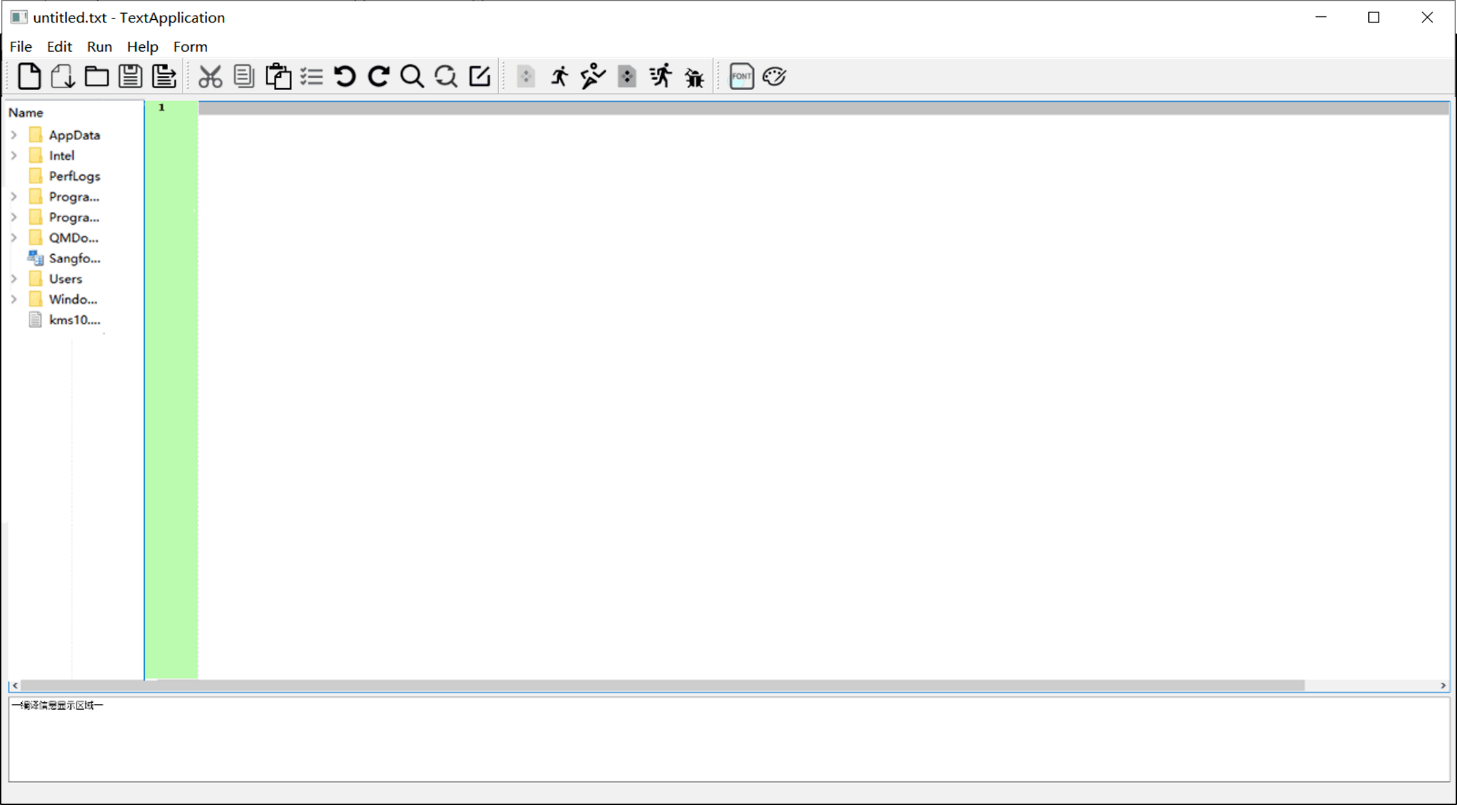
# 界面设计方案

## 界面整体布局设计方案

### 界面整体布局设计图



### 界面整体布局预览



## 标题栏设计方案



应用图标icon + 显示当前文件名（未保存时为untitled.txt）

## 菜单栏设计方案

### 菜单栏整体设计方案

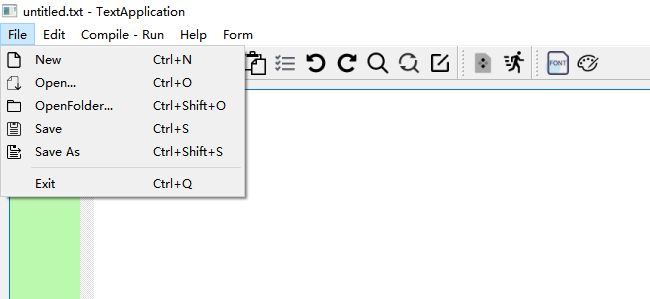


1. 文件选项
2. 编辑选项
3. 代码编译运行选项
4. 帮助选项

### 文件区域设计方案

文件区域实现功能：

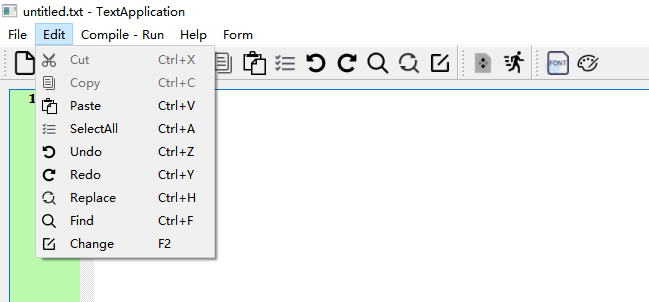
1. 新建文件 Ctrl + N
2. 打开文件 Ctrl + O
3. 打开文件夹 Ctrl + Shift + O
4. 保存 Ctrl + S
5. 另存为 Ctrl+ Shift + S
6. 退出程序 Ctrl + Q



### 编辑区域设计方案

编辑区域实现功能：

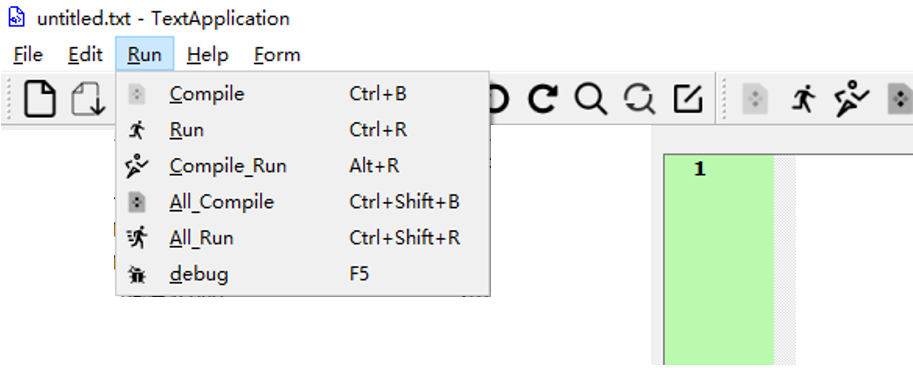
1. 复制 Ctrl + C
2. 粘贴 Ctrl + V
3. 剪切 Ctrl + X
4. 全选 Ctrl + A
5. 撤销 Ctrl + Z
6. 重做 Ctrl + Y
7. 查找 Ctrl + F
8. 替换 Ctrl + H
9. 变量名重命名 F2



### 编译运行区域设计方案

编译运行区域实现功能：

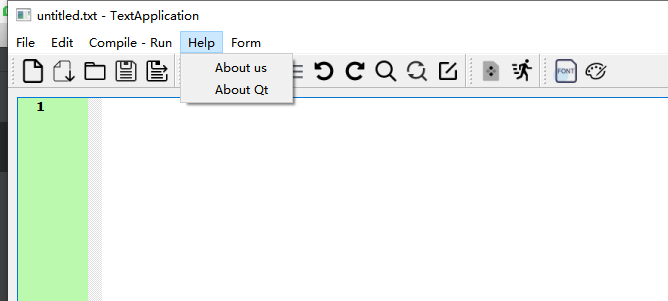
1. 编译 Ctrl + B
2. 运行 Ctrl + R
3. 编译运行 Alt + R
4. 多文件编译 Ctrl + Shift + B
5. 多文件运行 Ctrl + Shift + R
6. 调试 F5

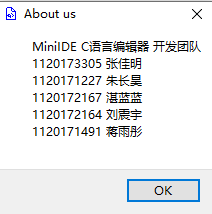


### 帮助区域设计方案

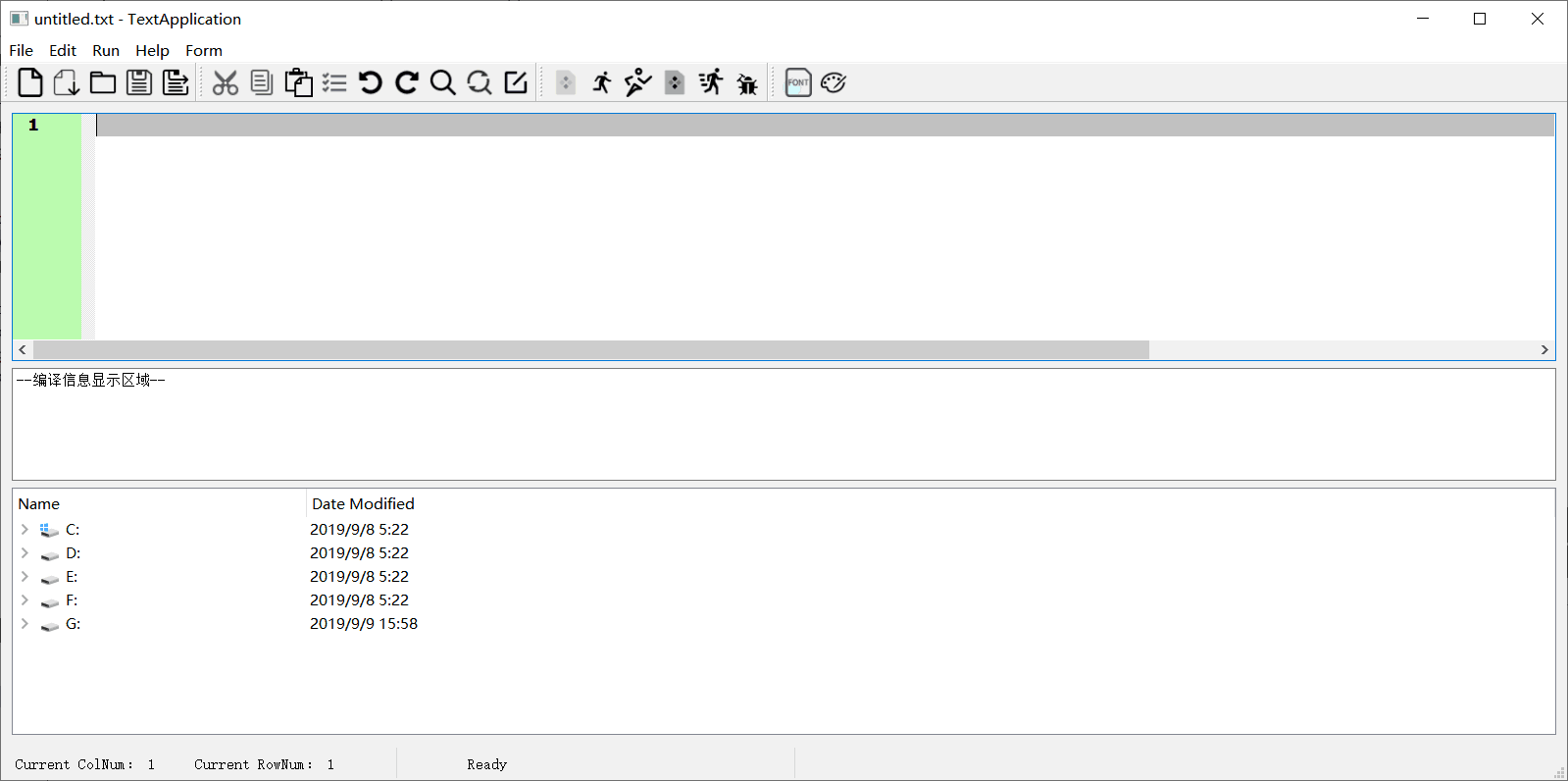
帮助区域实现功能：

1. 关于我们
2. 关于Qt



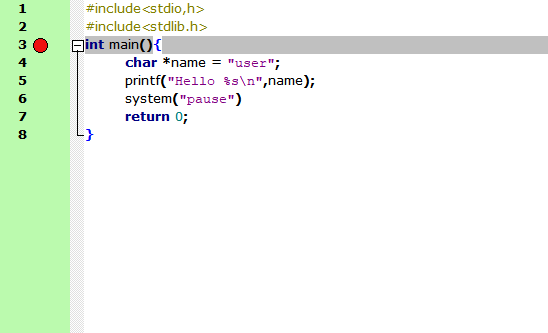


### 工具栏设计方案



1. 新建文件 Ctrl + N
2. 打开文件 Ctrl + O
3. 打开文件夹 Ctrl + Shift + O
4. 保存 Ctrl + S
5. 另存为 Ctrl+ Shift + S
6. 退出程序 Ctrl + Q
7. 复制 Ctrl + C
8. 粘贴 Ctrl + V
9. 剪切 Ctrl + X
10. 全选 Ctrl + A
11. 撤销 Ctrl + Z
12. 重做 Ctrl + Y
13. 查找 Ctrl + F
14. 替换 Ctrl + H
15. 变量名重命名 F2
16. 编译 Ctrl + B
17. 运行 Ctrl + R
18. 编译运行 Alt + R
19. 多文件编译 Ctrl + Shift + B
20. 多文件运行 Ctrl + Shift + R
21. 调试 F5

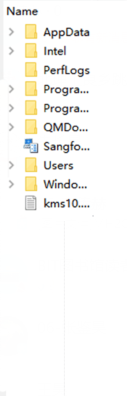
## 代码编辑区设计方案



代码编辑区实现功能：

1. 调整字体大小 Ctrl + mouseWheelEvent
2. 关键字高亮
3. 关键字补全
4. 括号匹配高亮
5. 括号、引号自动补全
6. 自动缩进
7. 代码自动整理
8. 代码整体整理 Ctrl + K + D
9. 代码块折叠
10. 多行代码注释 Ctrl + Shift + /
11. 行号显示
12. 当前行阴影
13. 添加断点
14. 注释隐藏 / 显示 F8

## 文件资源管理区设计方案

1. 文件资源管理区实现“打开文件夹”功能的可视化
2. 提供 可折叠展开的树形管理机制 和 “打开文件”功能的引用
3. 

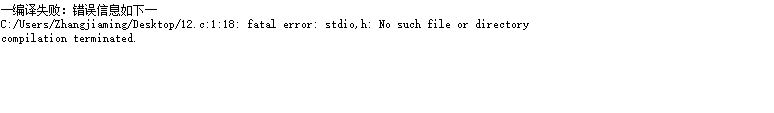
## 状态栏设计方案



状态栏提供功能：

1. 提供鼠标悬停所在功能的提示
2. 提供功能实现后的反馈提示
3. 提供编写时当前代码的行列显示

## 控制台设计方案



状态栏提供功能：

1. 提供当前文件编译结果的显示
2. 提供该程序运行的输入和输出

# 数据结构设计（UML设计）

## 各功能设计

### 编辑器

编辑器采用QsciScintilla类，词法分析器采用QsciLexerCPP类，可以实现关键字识别高亮、括号自动匹配高亮、自动排版、代码块折叠等功能。

### 编译功能

编译功能使用gcc。当用户点击编译按钮，系统会自动调用cmd里的gcc编译命令，通过gcc编译实现程序的编译，运行即直接运行编译好的.exe文件。

### 调试功能

调试窗口继承QDialog类，窗口的按钮对应写入的指令。

调试功能使用gdb。当用户点击调试按钮，系统首先调用编译功能，使用gcc对本文件进行编译。编译成功后，写入一个新线程进行调试。当用户点击不同的命令后，线程写入不同的指令进行调试，并解析gdb返回的信息显示到主窗口中。

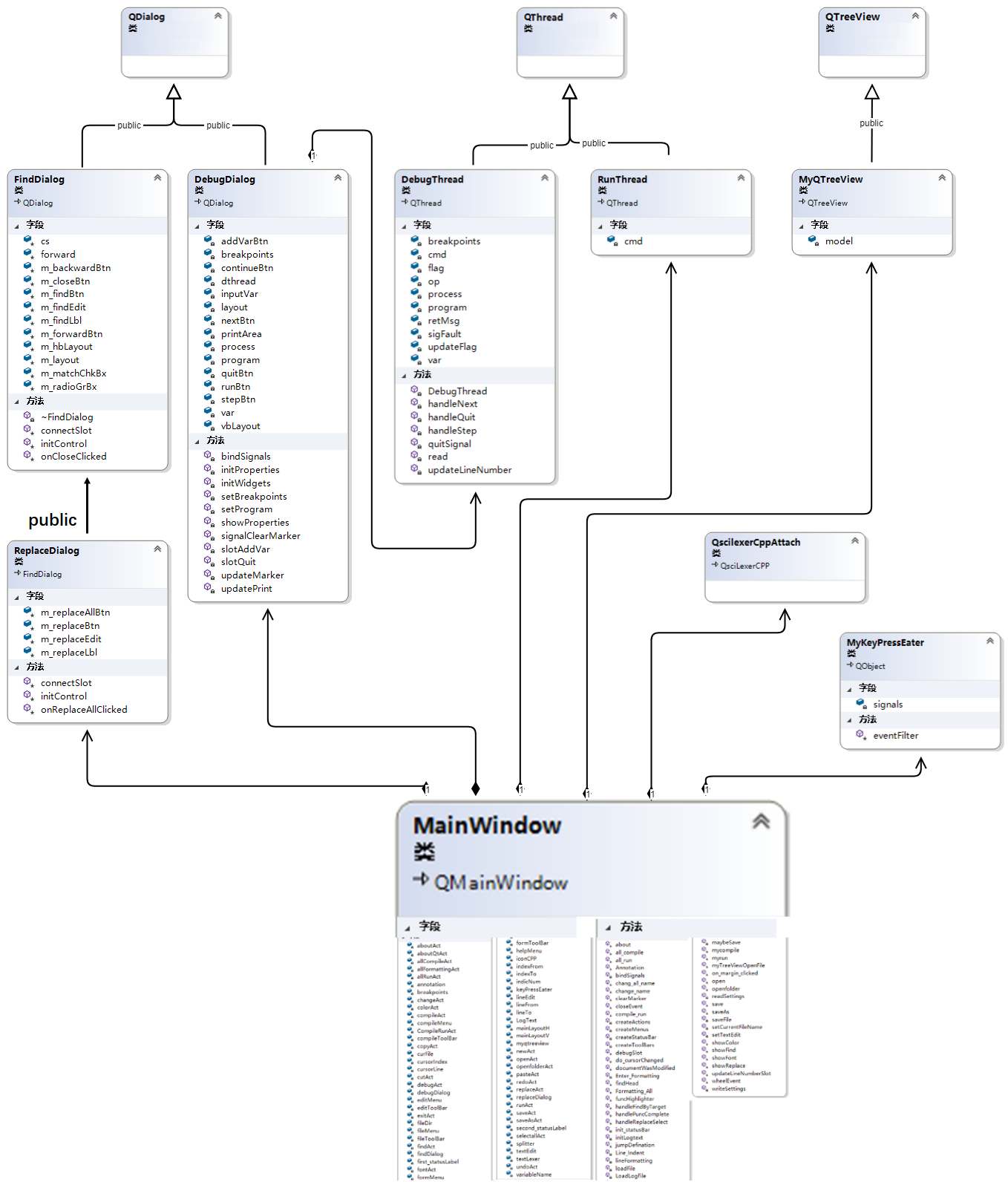
### 窗口功能

窗口方面，mainWIndow继承自QMainWindow类，findDialog继承自QDialog类，replaceDialog类继承自findDialog类。

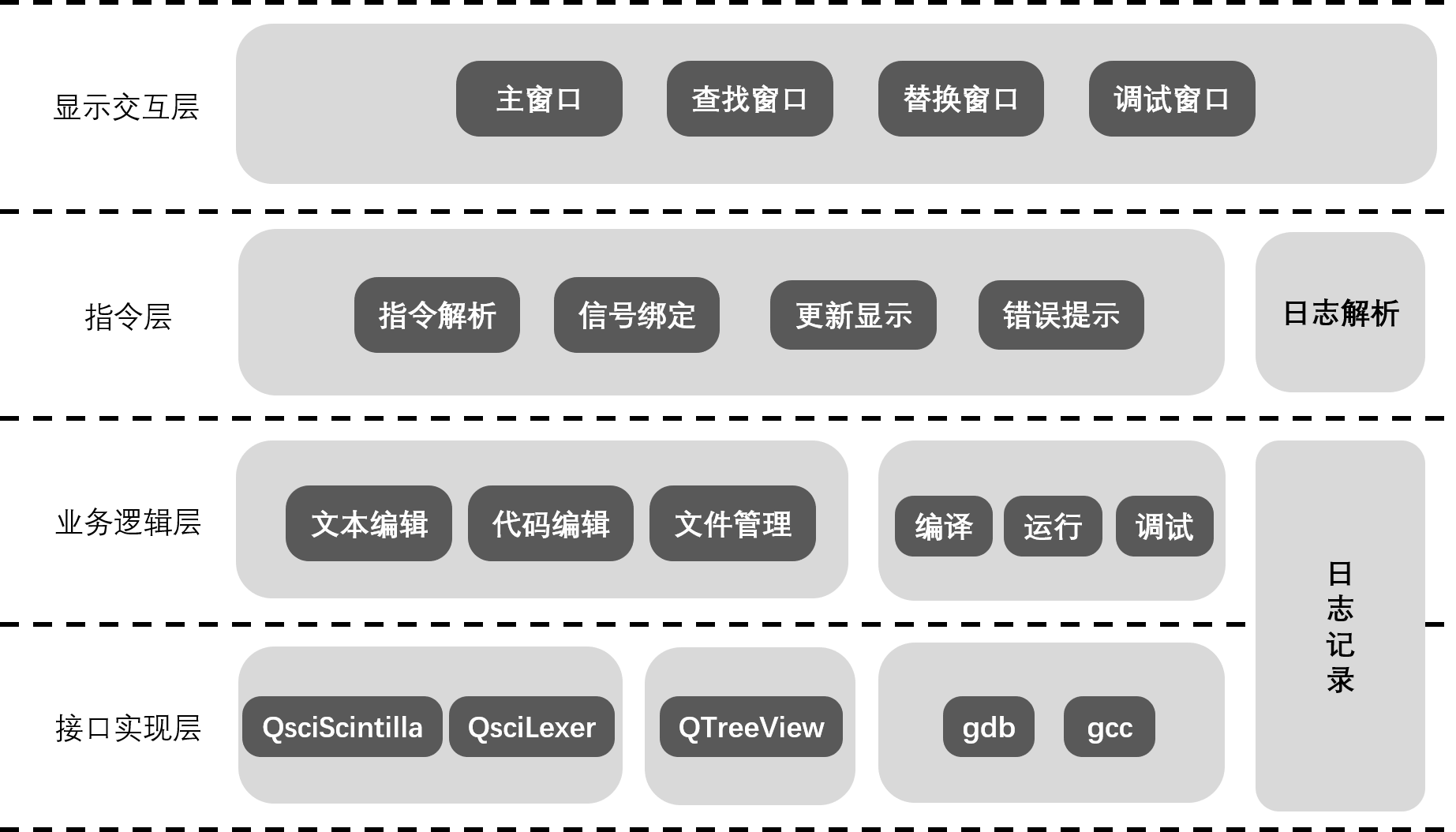
### 按钮监听功能

按键监听为自定义类，继承自QObject。

## UML图



# 系统结构图



# 程序功能表

|  |  |  |  |
| --- | --- | --- | --- |
| 程序功能表 | 文本编辑 | 复制 | ctrl + c |
| 粘贴 | ctrl + v |
| 查找 | ctrl + f |
| 替换 | ctrl + h |
| 撤销 | ctrl + z |
| 重做 | ctrl + y |
| 在状态栏显示当前行数 |  |
| 调整字体大小 | ctrl + wheelEvent |
| 代码编辑 | 关键字识别高亮 |  |
| 关键字联想提示、补全 | tab |
| 括号自动匹配高亮 |  |
| 括号自动补全 |  |
| 变量重命名 | F2 |
| 自动缩进 |  |
| 多行注释 | ctrl + shift + / |
| 代码块折叠 |  |
| 对已编辑的函数进行高亮显示 |  |
| 代码跳转 | ctrl + mouseClick |
| 行格式排版 |  |
| 整体格式排版 | F3 |
| 注释显示 / 隐藏 | F8 |
| 代码运行 | 通过gcc对当前文件编译 | ctrl + b |
| 通过gcc对当前文件运行 | ctrl + r |
| 通过gcc实现多文件编译 | ctrl + shift + b |
| 通过gcc实现项目运行 | alt + r |
| 在控制台显示编译结果 |  |
| 通过gdb对当前文件调试 | F5 |
| 文件管理 | 新建文件 | ctrl + n |
| 打开文件 | ctrl + o |
| 打开文件夹 | ctrl + shift + o |
| 保存文件 | ctrl + s |
| 另存为文件 | ctrl + shift + s |
| 多文件编辑 |  |
| 树形文件资源管理 |  |

# 关键算法设计

## 文本编辑模块

### 复制

1. 功能描述

复制用户在编辑区内选中的文本；

1. 快捷键：Ctrl + c
2. 实现机制

QsciScintilla库提供的槽函数

virtual void QsciScintilla ::***copy***();

### 粘贴

1. 功能描述

粘贴用户剪切板中复制的内容；

1. 快捷键：Ctrl + v
2. 实现机制

QsciScintilla库提供的槽函数

virtual void QsciScintilla ::***paste***();

### 查找

1. 功能描述

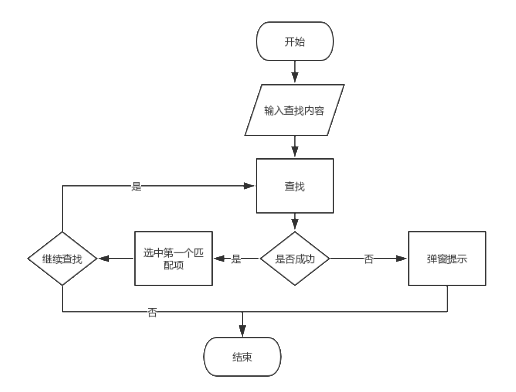
在文本编辑区内查找，可以选择全词/部分匹配、向前/向后查找。弹出对话框供用户输入待查找内容并设置查找选项。查找成功则在编辑区选中第一个符合条件的，失败则弹窗提示。

1. 快捷键：ctrl + f
2. 实现机制

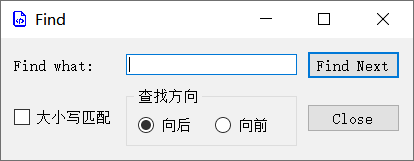
主窗口创建时即构建查找对话框，提供关键词输入区域和选项选择区域，绑定相关控件的槽函数。用户未触发前保持隐藏状态

触发查找后查找对话框状态被更新为显示，供用户输入待查找内容并设置查找选项。用户确认后弹窗对象将接收的查找要求以信号形式发送。主窗口编辑器接收信号后调用QsciScintilla::findFirst方法开始查找。查找成功选中查找项，查找失败则创建弹窗，提示用户查找失败。

1. 流程图



1. 窗口



### 替换

1. 功能描述

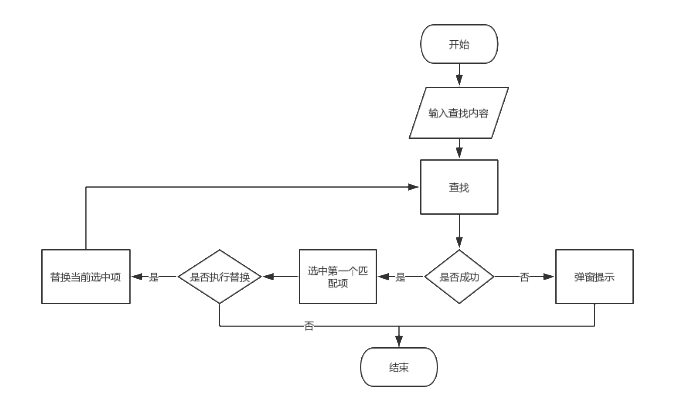
在文本编辑区内查找关键词并替换为新的关键词，替换选项包括替换选中/全部关键词，查找选项包括全词/部分匹配、向前/向后查找。弹出对话框供用户输入查找关键词、替换关键词并设置查找选项。查找成功则在编辑区选中第一个符合条件的，根据用户选择进行下一步操作，失败则弹窗提示。

1. 快捷键：ctrl + h
2. 实现机制

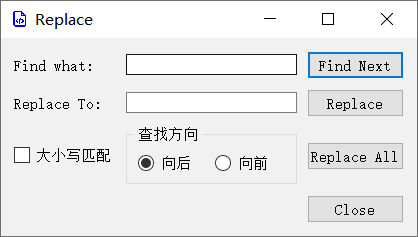
主窗口构建时即创建替换对话框。替换对话框类继承查找对话框类，继承查找对话框全部控件。此外，替换对话框提供替换词输入框。绑定所有部件的槽函数后替换对话框构建完毕，保持隐藏状态。

用户选择替换当前关键字时，调用QsciScintilla::replace方法替换关键字，随后调用QsciScintilla::findFirst方法定位到下一个关键字，等待用户操作。用户选择替换全部关键字时，循环上述替换流程，直至找不到待替换关键字。

1. 流程图



1. 窗口



### 撤销

1. 功能描述

撤销用户当前执行的操作；

1. 快捷键：Ctrl + z
2. 实现机制

QsciScintilla库提供的槽函数

virtual void QsciScintilla ::***undo***();

### 重做

1. 功能描述

重新执行用户撤销的操作；

1. 快捷键：Ctrl + y
2. 实现机制

QsciScintilla库提供的槽函数

virtual void QsciScintilla ::***redo***();

### 显示当前行数和列号

1. 功能描述

获取当前光标的行号和列号，并在状态栏显示；

1. 实现机制

将光标改变的信号与光标改变时间的槽函数进行绑定

connect(textEdit,SIGNAL(cursorPositionChanged(int,int)),this,SLOT(do\_cursorChanged()));

首先获取光标所在行号和列号

textEdit->getCursorPosition(&cursorLine,&cursorIndex);

将提示与行号列号拼接成为字符串

QString Tip = QString("Current ColNum： ") + QString::number(cursorLine)

+QString(“ “)

+ QString(" Current RowNum： ") + QString::number(cursorIndex);

将字符串显示在状态栏

first\_statusLabel->setText(Tip);

### 调整字体大小

1. 功能描述

通过“ctrl + wheelEvent”实现编辑区的字体大小调整；

1. 快捷键：ctrl + wheelEvent
2. 实现机制

重写鼠标滚轮事件wheelEvent

void MainWindow::***wheelEvent***(QWheelEvent \*event){

if(QApplication::keyboardModifiers () == Qt::ControlModifier){

if(event->delta()>0)//鼠标滚轮前转

textEdit->*zoomIn*();//放大

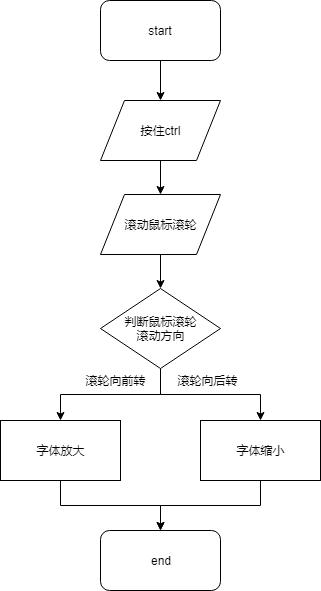
else

textEdit->*zoomOut*();//缩小

}

}

1. 流程图



## 代码编辑模块

### 关键字识别高亮

1. 功能描述

检测编辑区文本是否为关键字，如果是则高亮显示

1. 实现机制

/\* 通过语法分析器QsciLexer \*/

QsciLexer \*textLexer;

绑定C++的关键字

textLexer = new QscilexerCppAttach;

设置自带的注释行为绿色

textLexer->*setColor*(QColor(Qt:: green),QsciLexerCPP::CommentLine);

设置自定义关键字的颜色为黄色

textLexer->*setColor*(QColor(Qt:: yellow),QsciLexerCPP::KeywordSet2);

将语法分析器绑定在编辑器上

textEdit->*setLexer*(textLexer);

### 关键字联想提示、补全

1. 功能描述

根据用户当前输入提供关键字提示列表，按Tab进行补全；

1. 实现机制

/\* 通过语法分析器QsciLexer \*/

QsciLexer \*textLexer;

绑定C++的关键字

textLexer = new QscilexerCppAttach;

将语法分析器绑定在编辑器上

textEdit->*setLexer*(textLexer);

设置开启自动补全功能

textEdit->setAutoCompletionFillupsEnabled(true);

### 括号自动匹配高亮

1. 功能描述

查找当前光标所在括号的所匹配的括号，并通过高亮显示；

1. 实现机制

通过QsciScintilla自带的库函数

textEdit->*setBraceMatching*(QsciScintilla::SloppyBraceMatch);

### 符号自动补全

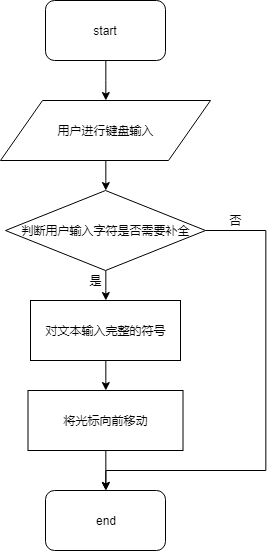
1. 功能描述

根据用户输入的符号进行符号补全，并将光标移动至符号中间；

1. 实现机制

监听键盘输入，当键盘输入**(**,**[**,**{**等符号时，输入文本为**()**,**[]**,**{}**，并向前移动光标；

1. 流程图



### 变量重命名

1. 功能描述

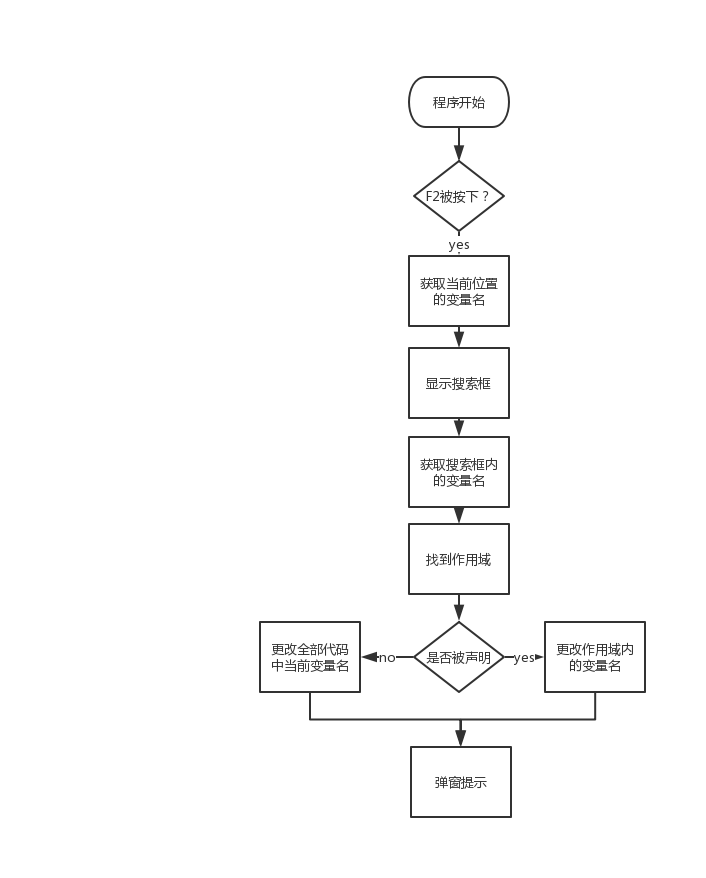
将光标位于需要更改的变量上，更改变量所在作用域的名字。

1. 快捷键：F2
2. 实现机制

正则表达式筛选出“类型名 变量名”的文本，获取这样的变量名相同的文本位置A和B，且保证需要更改的变量名在A和B之间。

使用Qscintilla的findFirst和findFirstInSelection，配合replace更改变量名称。

1. 流程图



### 自动缩进

1. 功能描述

当监听到键盘输入回车时，对此行进行缩进整理。具体实现缩进的情况有以下几个方面：

* 正常情况下，下一行的缩进与上一行保持一致
* 如果回车前一个符号为{，则下一行的缩进要在正常基础上多一次
* }所在的行缩进与其对应的{保持一致

1. 实现机制

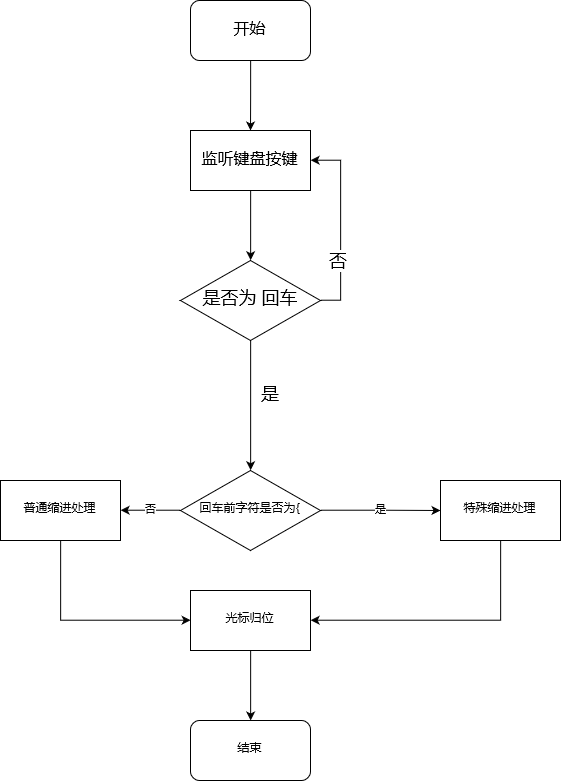
当keyPressSiganl\_puncComplete(int)信号监听到键盘输入回车时，对应的槽函数中将调用Enter\_Formatting（）函数，此函数专门进行回车相关的规范化处理。

在函数中，首先要判断回车前的符号是否为 { 。如果不是，则调用Line\_Indent(int linenum,int flag)，同时将flag置为0 ，以进行普通的缩进处理；否则调用相同函数，但将flag置为1，以进行额外缩进处理。

在缩进函数的执行内部，首先是对上一行的tab数量进行检测并记录，同时需要考虑flag的影响，得出本行需要的tab数。之后再对本行已有的tab数进行统计，相减后即得出实际需要添加的tab数，再进行添加操作即可完成缩进。

最后，由于进行了较多检测、添加操作，最后需要根据修改情况将光标复位到合适的地方。此处我使用了tab\_num记录添加空格情况，之后使用setCursorPosition进行复位。

1. 流程图



### 多行注释

1. 功能描述

将用户选择文本区域逐行通过添加 // 进行注释；

1. 快捷键：Ctrl + Shift + /
2. 实现机制

首先获取用户选中区域的起始行号、末尾行号

textEdit->getSelection(&lineFrom,&indexFrom,&lineTo,&indexTo);

再判断此时进行注释行为还是取消注释行为

通过逐行判断开头字符是否为””来判断下一步事件

如果是注释行为，在每一行前插入””进行注释

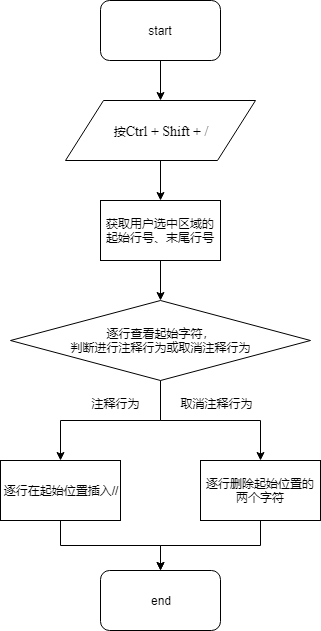
textEdit->*insertAt* (tr(""), i, 0 );

如果是取消注释行为，将每一行的前两个字符删除

textEdit->*setSelection*(i,0,i,2);

textEdit->*removeSelectedText*();

1. 流程图



### 注释隐藏 / 显示

1. 功能描述

展示/隐藏当前编辑区内所有注释。

1. 快捷键：F8
2. 数据结构

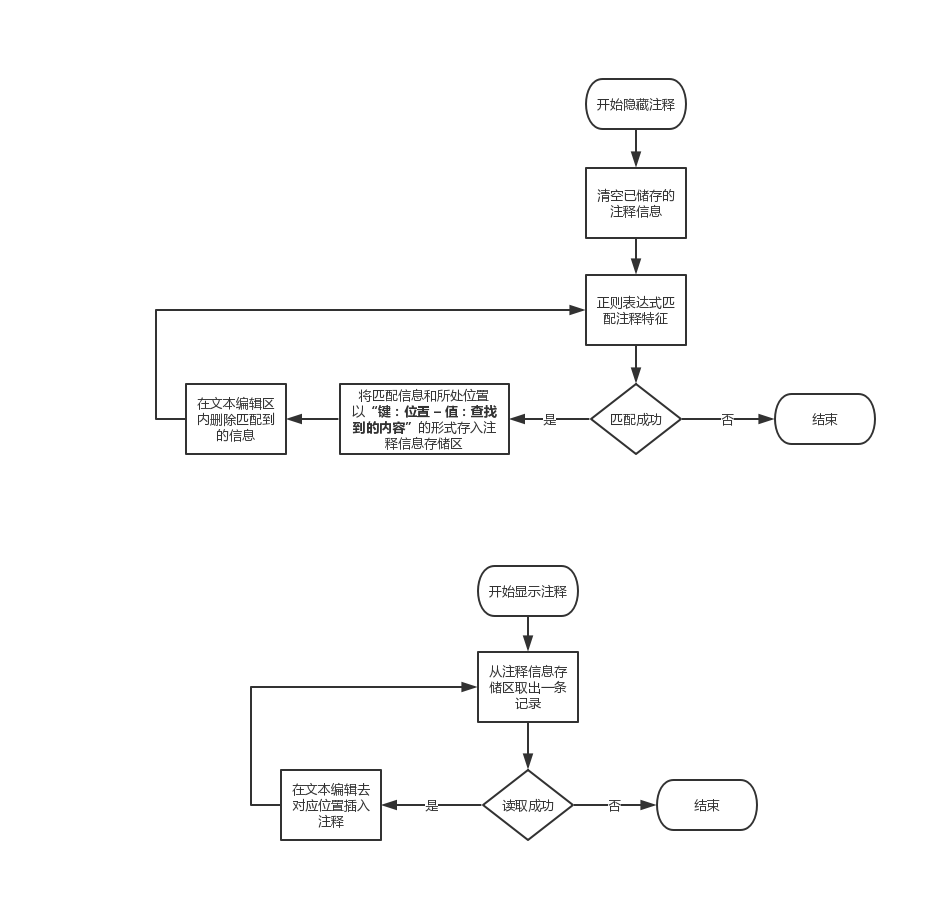
map，键值对为“键：行号 – 值：注释”

1. 实现机制

隐藏注释：获取文本编辑区内文本内容，使用正则表达式查找“//”和“/\*\*/”注释。查找成功后将查找到的内容以“键：行号 – 值：查找到的内容”的形式存入map中，同时在文本编辑区内删除选中的内容。循环此过程直至没有匹配。

显示注释：从map中获取一个键值对，在当前文本编辑区内指定行号插入注释。循环此过程直至遍历map中 所有键值对。

1. 流程图



### 代码块折叠

1. 功能描述

在编辑区左侧设置按钮，能够将代码进行折叠；

1. 实现机制

virtual void ***setFolding***(FoldStyle fold, int margin = 2);

### 对已编辑的函数进行高亮显示

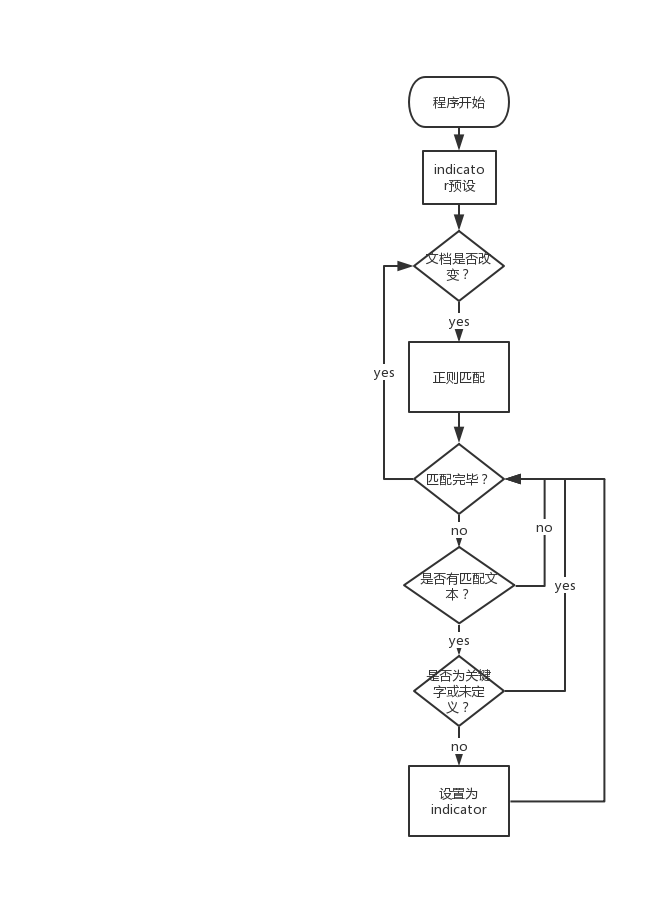
1. 功能描述

对于已经有了声明的函数会进行高亮显示，未声明的函数不会高亮。

1. 实现机制

采用qscintilla的indicator部件，当非c语言的关键字后出现括号时即可判定这是一个函数。利用正则匹配全部代码中这样的文本，将他们设定为一个indicator，并进行style设置即可完成高亮。

1. 流程图



### 代码跳转

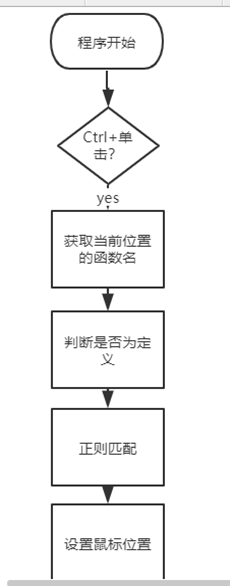
1. 功能描述

CTRL + 双击一个函数的声明或者具体引用时，会自动跳到他的定义；CTRL+单击一个函数的定义时，会跳转到他的声明。

1. 快捷键：ctrl + 双击
2. 实现机制

正则表达式匹配函数名字，由于声明肯定在代码的最前面，所以第一个匹配到的一定是声明，而定义时，函数名所在行要么一定有大括号，要么没有分号，根据这个匹配到的一定是函数定义。再获取这两个位置，判断点击的是否为定义，使用scintilla的setcursorPosition函数即可实现跳转。

1. 流程图



### 行格式排版

1. 功能描述

当监听到键盘输入“;”时，对此行进行格式化整理。具体实现的格式化内容有以下几个方面：

* 对于"+"、"-"、"\*"、"/"、"("、")"、"="、"<"、">"、","、”;” 这些符号，在符号的两侧使用空格进行分隔
* 一些特殊的符号组保留原有的格式，如：”++” , “--”, “->” , “+=”, “-=”等等

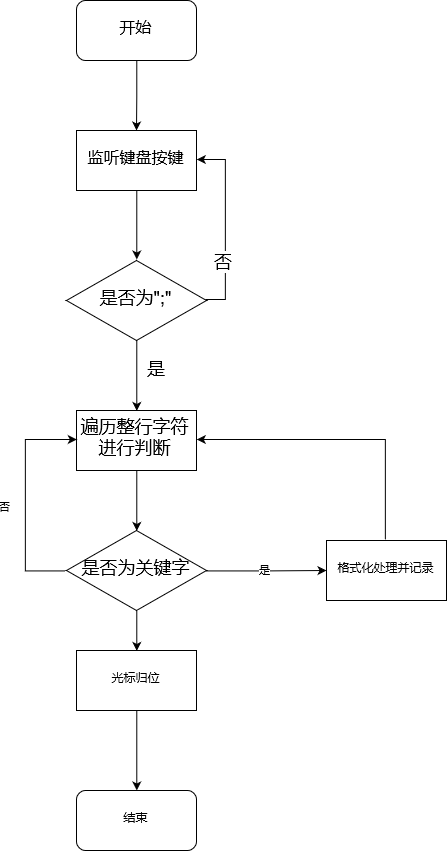
1. 实现机制

当keyPressSiganl\_puncComplete(int)信号监听到键盘输入";"时，对应的槽函数中将调用lineFormatting()函数，同时将当行行号作为参数传入函数中。

在函数中，首先创建一个参照表：key\_word\_table，用来存储上述的关键字。然后获取当前行的字符数量，接下来就进行遍历，将每个字符与参照表进行对比，判断是否为需要格式化的字符。同时，由于多个关键字相邻时不能重复添加空格，以及 ++、->、+= 等固定组合中间不能有空格，因此在添加空格前加入对以上情况的判断即可。

最后，由于进行了较多检测、添加操作，最后需要根据修改情况将光标复位到合适的地方。此处我使用了indexnum记录添加空格情况，之后使用setCursorPosition进行复位。

1. 流程图



### 整体格式排版

1. 功能描述

当收到 formatting all的信号时，对已编辑的全部代码进行格式化和缩进整理。具体实现的格式化内容有以下几个方面：

* 对于"+"、"-"、"\*"、"/"、"("、")"、"="、"<"、">"、","、”;” 这些符号，在符号的两侧使用空格进行分隔
* 一些特殊的符号组保留原有的格式，如：”++” , “--”, “->” , “+=”, “-=”等等
* 行首保持相同缩进，如果有{}的嵌套则进行额外缩进

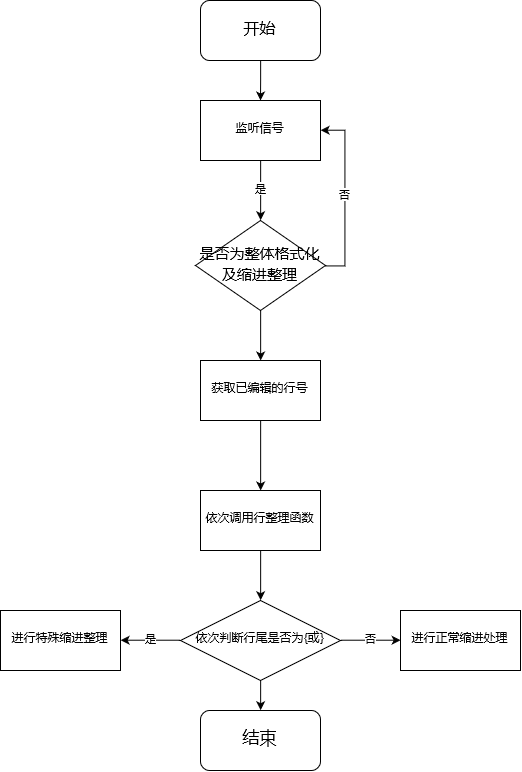
1. 快捷键：F3
2. 实现机制

当收到全体格式化和缩进整理的信号时（输入快捷键或鼠标点击对应图标），将调用对应的槽函数Formatting\_All（），对已编辑的全部内容进行格式化和缩进整理。

在函数中，首先需要获得已编辑的代码的行数。接下来就进行遍历，对每行调用行格式化代码lineFormatting（），以实现全体代码的格式化整理。之后需要再遍历一次，对每行分别检测本行行尾是否为 { ,以及下行结尾是否为 } ,分情况调用Line\_Indent（）函数进行缩进规范，从而实现对整体代码的缩进规范整理。

行格式化函数与行规范函数的实现机制可参考其专门的模块。

1. 流程图



## 编译运行模块

### 通过gcc对当前文件编译

1. 功能描述

调用gcc编译器在当前文件路径下生成可执行.exe文件；

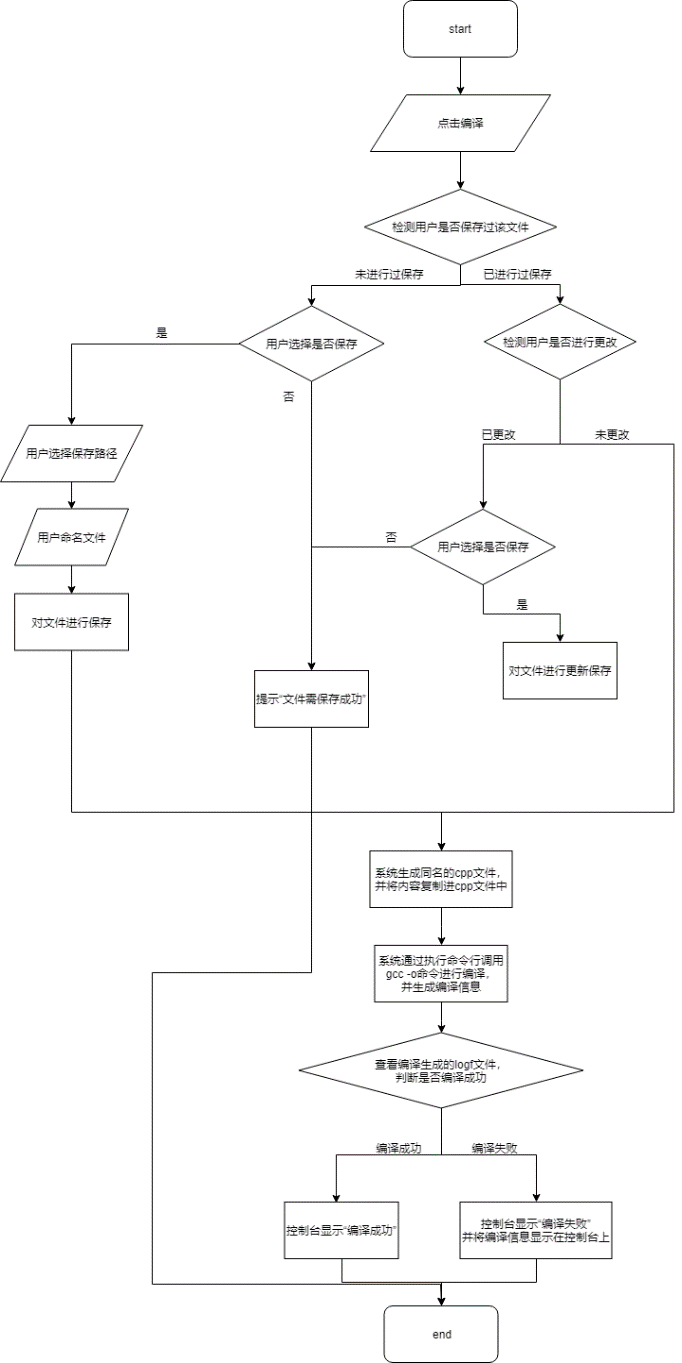
1. 快捷键：Ctrl + B
2. 实现机制

通过命令行执行调用gcc语句

const char \*s = filename.toStdString().data();

cmd.sprintf("gcc -o %s.exe %s.c 2>%s.log -g",s,s,s);

1. 流程图



### 通过gcc对当前文件运行

1. 功能描述

运行文件编译成功后生成的exe文件；

1. 快捷键：Ctrl + r
2. 实现机制

首先判断是否编译成功

if(LoadLogFile(LOG+".log"))

若成功，新建线程，在此线程通过命令行执行exe文件

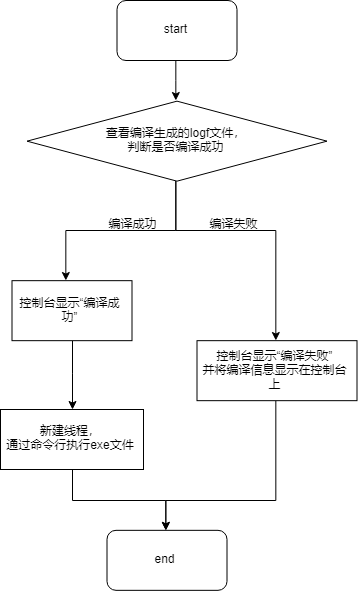
QString cmd = curFile + ".exe ";

RunThread \*rthread = new RunThread(cmd);

connect(rthread, &RunThread::finished, rthread, &QObject::deleteLater);

rthread->start();

1. 流程图



### 通过gcc实现多文件编译

1. 功能描述

查看主文件引用的外部头文件，并调用gcc编译器将其同主文件在当前文件路径生成可执行.exe文件；

1. 快捷键：Ctrl + Shift + B
2. 实现机制

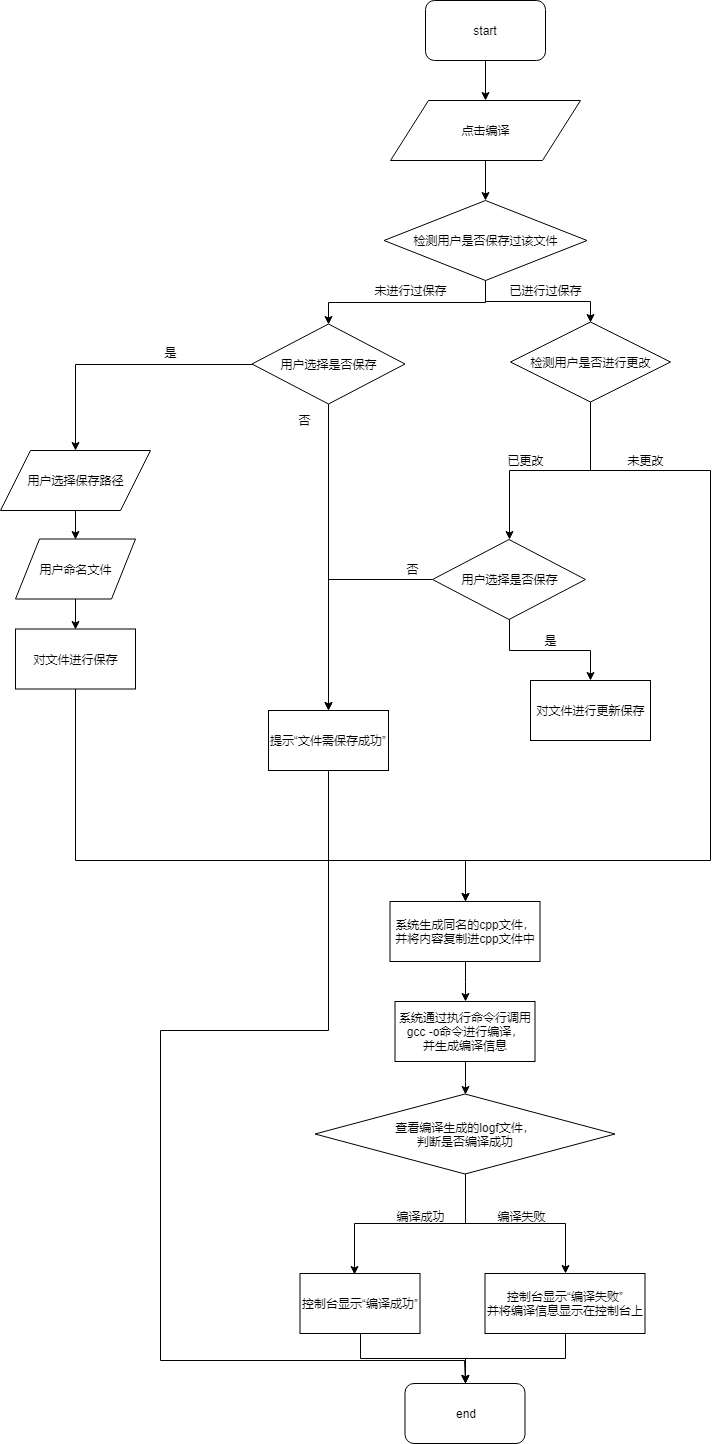
首先查看该文件通过#include” ”引用的外部头文件

QList<QString> headList = findHead()

将当前文件与头文件对应的源文件拼接生成字符串命令

通过执行命令行调用g++执行编译

1. 流程图



### 通过gcc实现项目运行

1. 功能描述

运行文件编译成功后生成的exe文件；

1. 快捷键：Ctrl + Shift + R
2. 实现机制

首先判断是否编译成功

if(LoadLogFile(LOG+".log"))

若成功，新建线程，在此线程通过命令行执行exe文件

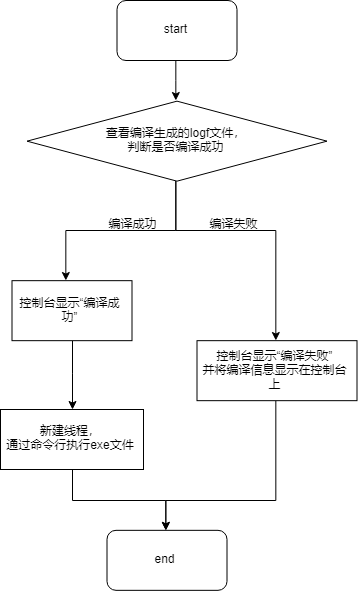
QString cmd = curFile + ".exe ";

RunThread \*rthread = new RunThread(cmd);

connect(rthread, &RunThread::finished, rthread, &QObject::deleteLater);

rthread->start();

1. 流程图



### 在控制台显示编译结果

1. 功能描述

将编译结果显示在控制台；

1. 实现机制

编译时生成与代码文件同名的log文件，存储编译生成信息

cmd.sprintf("gcc -o %s.exe %s.c 2>%s.log -g",s,s,s);

读取log文件的内容

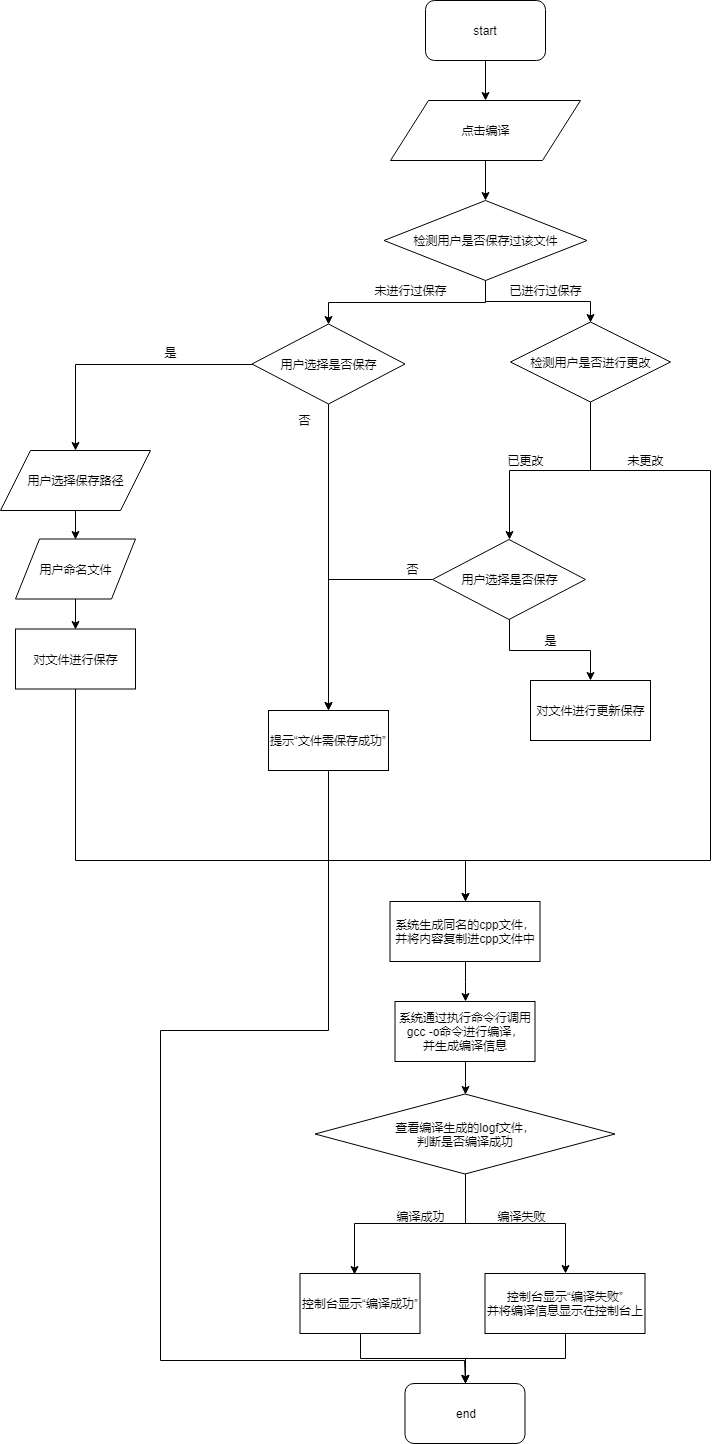
QTextStream in(&file);

QString logInfo = in.readAll();

若内容为空，则说明编译成功，显示编译成功；若内容不为空，说明编译失败，显示编译信息

将编译结果显示在控制台

1. 流程图



### 通过gdb对当前文件调试

1. 功能描述

编译当前代码，并启动gdb追踪代码的运行流程。将程序运行位置显示在编辑区侧边栏，将程序运行状态显示在调试对话框中。在程序运行过程中启用Debug模式，通过设置断点、添加查看变量可以分析定位异常发生的位置，以及追踪在运行过程中参数的变化。

提供的功能如下：

* 添加查看：接收用户输入的变量。在将来的调试过程中随语句执行显示设置的变量在当前语境的值；
* 开始调试：根据设置的断点呵查看变量信息开启gdb环境，进行调试。调试过程中不允许更改查看变量设置；
* 单步进入：执行下一条语句。如果该语句为函数调用，则进入该函数执行其中的第一条语句；
* 单步跳过：执行下一条语句。如果该语句为函数调用。则不会进入函数内部执行（即不会一步步地调试函数内部语句）；
* 继续执行：继续运行程序，直到遇到下一个断点；
* 退出调试：强制终止当前调试过程，退出gdb环境。

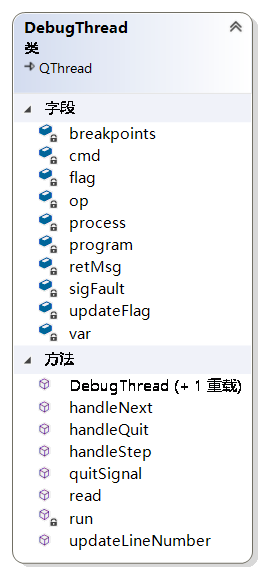
1. 快捷键：F5
2. 实现机制

用户执行调试选项后，执行MainWindow::mycompile()进行编译。编译完成后调用DebugDialog::show()显示调试窗口，接收用户设置的查看变量信息，接收用户通过点击编辑区边栏设置的断点信息。设置成功，开启DebugThread线程，通过QProcess运行gdb开始调试。

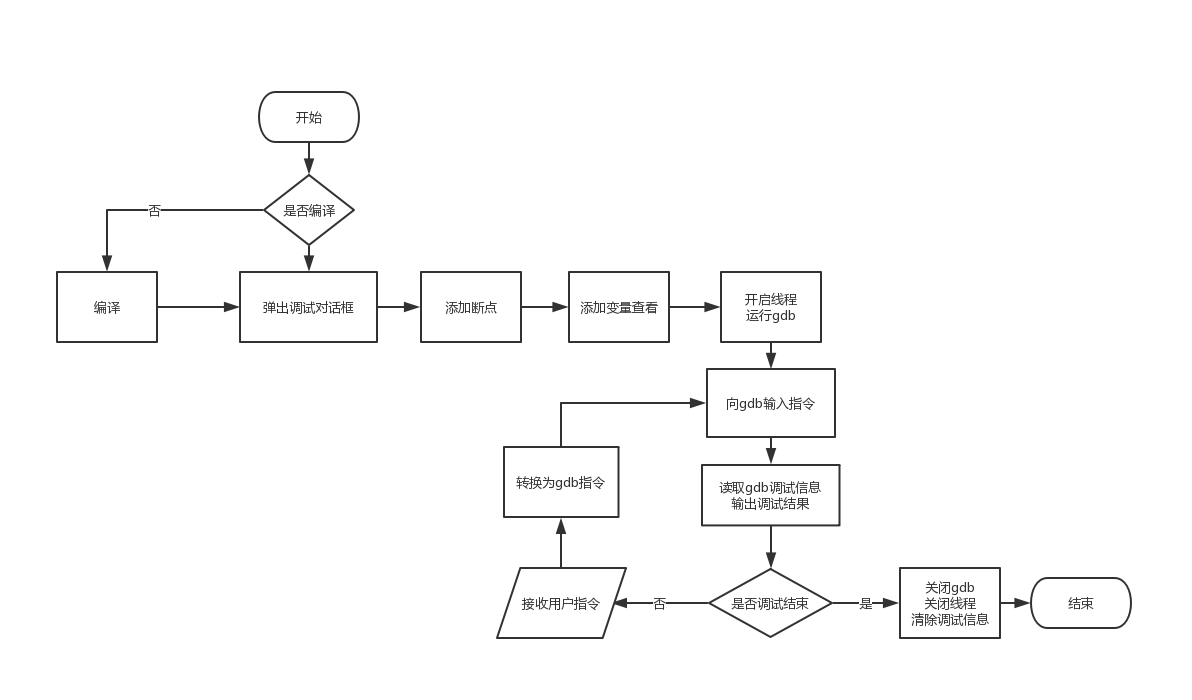
调试过程中，通过DebugDialog接收用户操作指令，讲指令转换为gdb指令通过QProcess::write()传递给gdb程序。从gdb标准输出和标准错误文件中读取gdb调试信息，转换，通过DebugThread::updateLineNumber信号发送给MainWindow更新程序运行位置；通过DebugThread::updateSignal信号向DebugDialog更新程序运行状态。循环此过程直至调试结束。

调试结束后，通过DebugThread::quitSignal信号更新MainWindow和DebugDialog，通知用户调试结束。随后结束关闭QProcess调用的gdb，结束DebugThread线程。

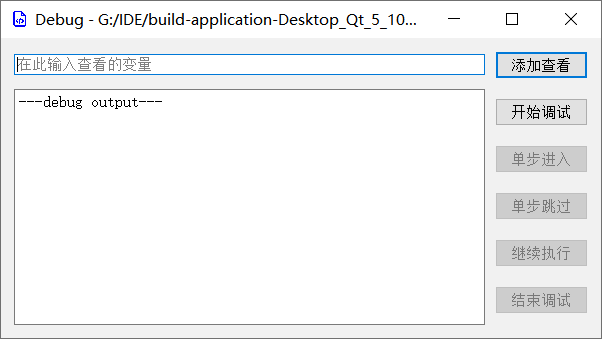
1. 类图



1. 流程图



1. 窗口



## 文件管理模块

### 新建文件

1. 功能描述

在编辑区新建空白文件编辑区（检测当前文件是否保存）；

1. 快捷键：Ctrl + N
2. 实现机制

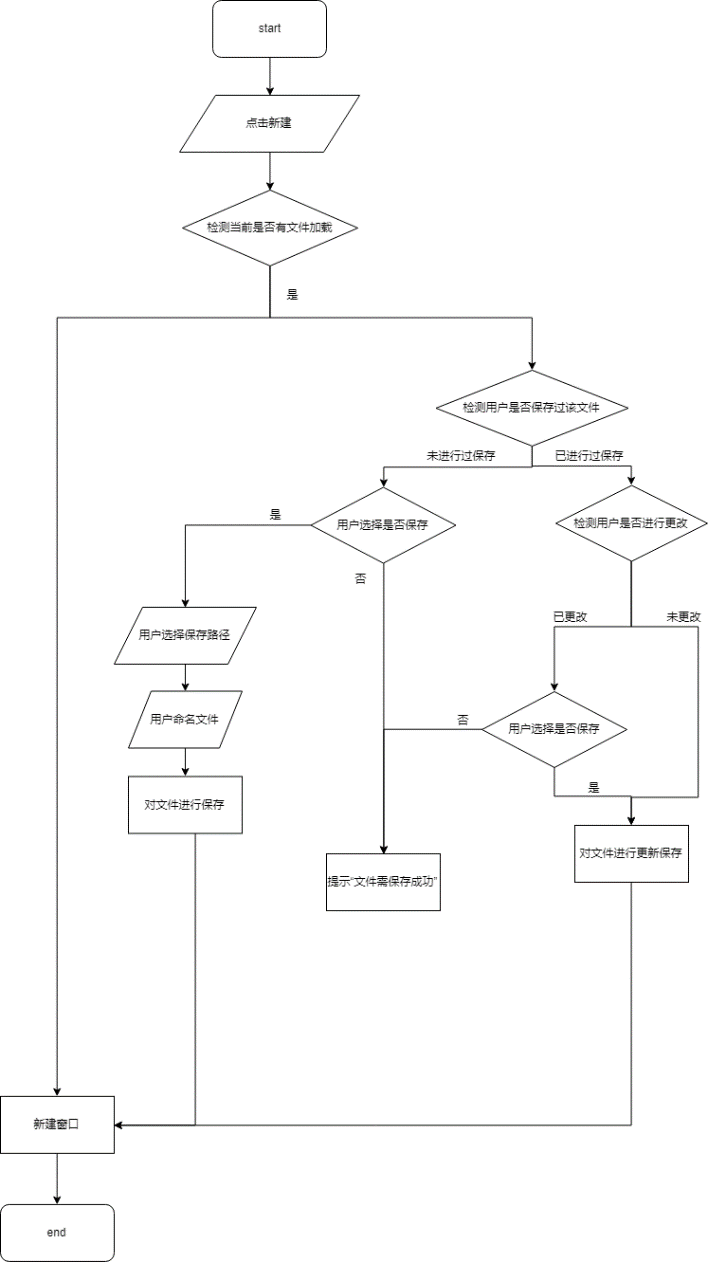
首先查看当前文件是否需要进行保存

然后在当前新建一个窗口生成新文件

textEdit->*clear*();

setCurrentFileName("");

1. 流程图



### 打开文件

1. 功能描述

用户通过弹窗选择打开的文件，并将文件加载在文件编辑区

1. 快捷键：Ctrl + o
2. 实现机制

首先查看当前文件是否需要进行保存

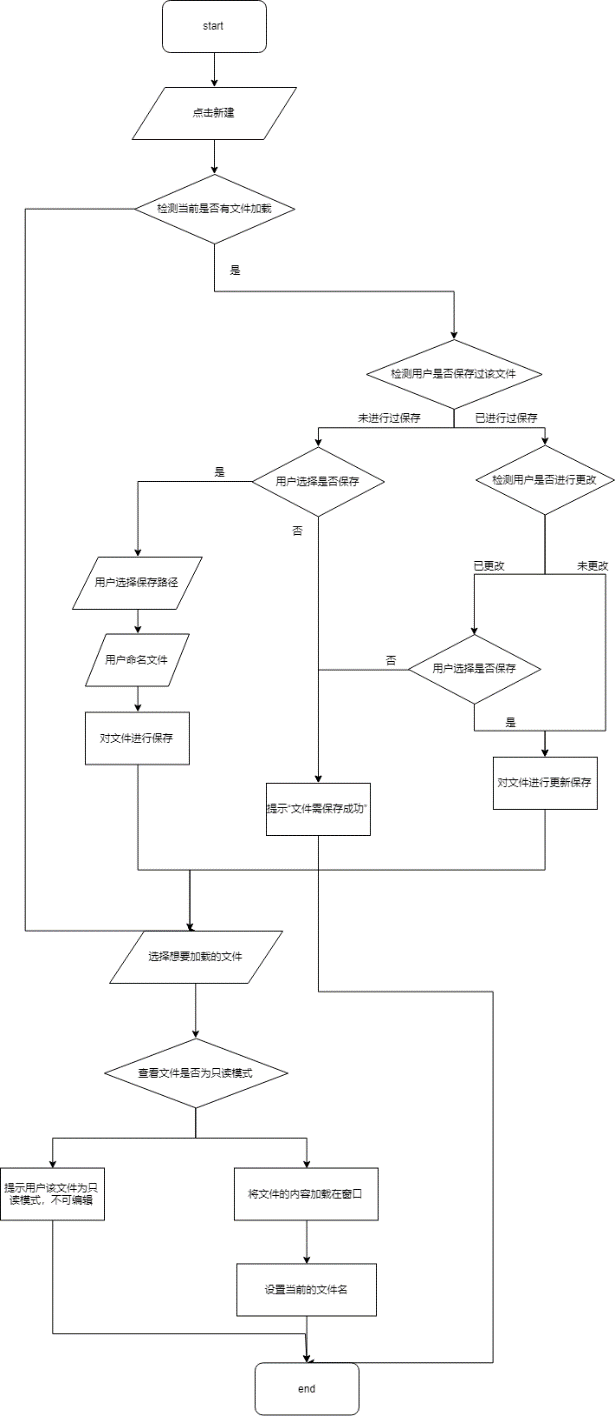
再将选择文件的内容加载在窗口、并设置当前文件名

QTextStream File\_in(&file);

textEdit->*setText*(File\_in.readAll());

setCurrentFileName(fileName);

1. 流程图



### 打开文件夹

1. 功能描述

用户通过弹窗选择想要打开的文件夹，并将文件夹展现在树形文件资源管理区

1. 快捷键：Ctrl + Shift + O
2. 实现机制

首先用户选择想要打开的文件夹

fileDir = QFileDialog::getExistingDirectory(this);

重置树形文件资源管理的根目录，并将其设置为用户选择文件夹的路径

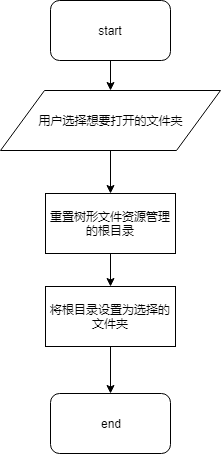
myqtreeview->*reset*();

myqtreeview->model = new QDirModel;

myqtreeview->*setModel*(myqtreeview->model);

myqtreeview->*setRootIndex*(myqtreeview->model->index(fileDir));

1. 流程图



### 保存文件

1. 功能描述

用户通过弹窗选择想要保存的文件夹，并将当前文件保存

1. 快捷键：Ctrl + s
2. 实现机制

首先查看本文件是否进行过保存

若未进行过保存，则提示用户选择保存路径

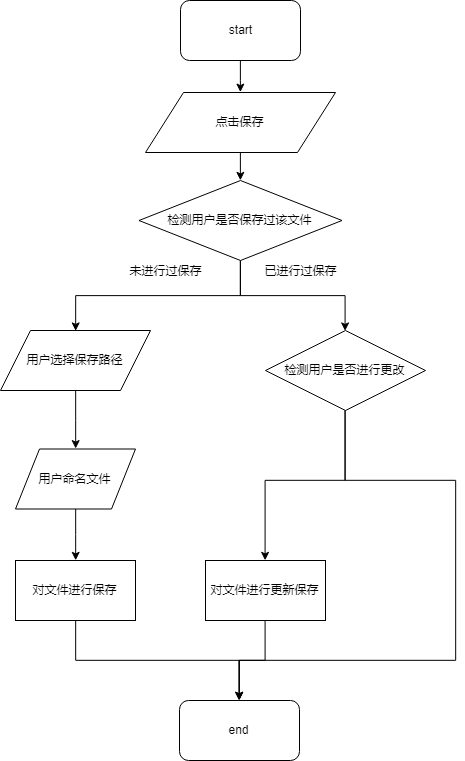
QString fileName = QFileDialog::getSaveFileName(this);

将当前文件进行保存

QTextStream File\_out(&file);

File\_out << textEdit->text();

1. 流程图



### 另存为文件

1. 功能描述

用户通过弹窗选择想要保存的文件夹，并将当前文件保存；

1. 快捷键：Ctrl + Shift + S
2. 实现机制

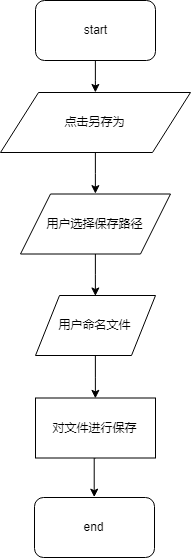
用户选择想要保存的路径，并填写文件名

再保存文件

QTextStream File\_out(&file);

File\_out << textEdit->text();

1. 流程图



### 多文件编辑

1. 功能描述

通过多个窗口实现多文件编辑

1. 实现机制

在树形目录资源管理中双击打开的文件

若本窗口没有已经加载的文件（如果现在文件名是空且没被更新就本窗口打开），在本窗口加载文件

if(curFile==""&&!textEdit->isModified())

loadFile(myqtreeview->selectFilePath);

否则打开新窗口，并在新窗口加载文件加载文件，并将文件名、根目录闯入新窗口中

MainWindow \*newMainWindow = new MainWindow();

newMainWindow->myqtreeview->*reset*();

newMainWindow->myqtreeview->model = new QDirModel;

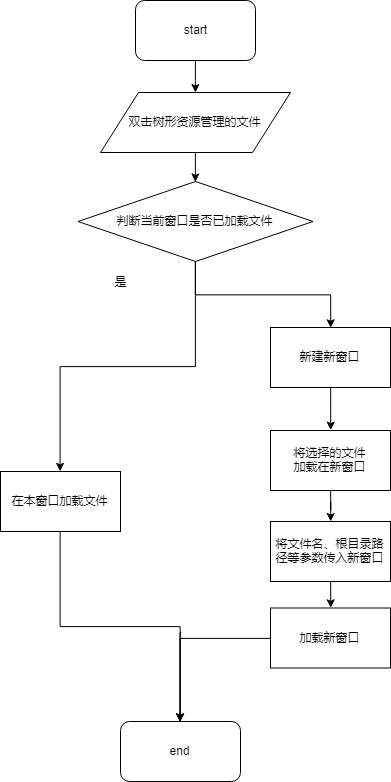
newMainWindow->myqtreeview->*setModel*(myqtreeview->model);

newMainWindow->myqtreeview->*setRootIndex*(myqtreeview->model->index(fileDir))

newMainWindow->loadFile(myqtreeview->selectFilePath);

newMainWindow->show();

1. 流程图



### 树形文件资源管理

1. 功能描述

将用户选择打开的文件夹进行展示；

1. 实现机制

获取用户点击文件的文件名

逐步向上获取其父节点的名，并进行拼接

拼接为完整路径后，获得点击文件的绝对路径

myqtreeview->selectFilePath = QString(index.data().toString()) + myqtreeview->selectFilePath;

while(index.parent().data().toString()!=""){

index=index.parent();

myqtreeview->selectFilePath = QString(index.data().toString()) + QString("/") + myqtreeview->selectFilePath;

}

进行多窗口编辑

1. 流程图

