

NCNDiDE—C语言编辑器

软件系统设计文档



2020-9-25

阿巴阿巴怪小组

v 1.9

目录

[1. 整体架构设计 2](#_Toc52054409)

[2.数据结构设计&算法设计 3](#_Toc52054410)

[2.1字符串编辑功能 3](#_Toc52054411)

[2.1.1插入 3](#_Toc52054412)

[2.1.2删除 3](#_Toc52054413)

[2.1.3替换 4](#_Toc52054414)

[2.2文本块编辑功能 5](#_Toc52054415)

[2.2.1复制 5](#_Toc52054416)

[2.2.2粘贴 5](#_Toc52054417)

[2.2.3剪切 6](#_Toc52054418)

[2.3窗口显示功能 6](#_Toc52054419)

[2.3.1光标移动 6](#_Toc52054420)

[2.3.2行号设置与跳转 6](#_Toc52054421)

[2.3.3关键字识别 6](#_Toc52054422)

[2.3.4括号匹配与补全 7](#_Toc52054423)

[2.3.5自动排版 7](#_Toc52054424)

[2.4文件资源管理功能 7](#_Toc52054425)

[2.4.1新建文件 7](#_Toc52054426)

[2.4.2打开文件 8](#_Toc52054427)

[2.4.3保存文件 9](#_Toc52054428)

[2.4.4另存文件 10](#_Toc52054429)

[2.4.5文件资源管理区 11](#_Toc52054430)

[2.5调试编译运行功能 11](#_Toc52054431)

[2.5.1调试功能 11](#_Toc52054432)

[2.5.2编译功能 12](#_Toc52054433)

[2.5.3运行功能 13](#_Toc52054434)

[3.UML图示分析 13](#_Toc52054435)

[3.1用例图 13](#_Toc52054436)

[3.2类图 14](#_Toc52054437)

[3.3活动图 15](#_Toc52054438)

[3.4状态图 15](#_Toc52054439)

# 整体架构设计

# 2.数据结构设计&算法设计

## 2.1字符串编辑功能

### 2.1.1插入

由类basicEdit实现，继承自QTextEdit类，直接引用QTextEdit中set等多功能实现。

**流程图：**

图片包含 游戏机

描述已自动生成

### 2.1.2删除

由类basicEdit实现，继承自QTextEdit类，直接引用QTextEdit中set等多功能实现。

**流程图：**

图片包含 游戏机

描述已自动生成

### 2.1.3替换

继承QDialog，通过textLine中的find方法进行查找，最后用QString进行文本对应的替换。具体如图：



**流程图：**

图片包含 游戏机

描述已自动生成

## 2.2文本块编辑功能

### 2.2.1复制

QsciScintilla库提供的槽函数 virtual void QsciScintilla::copy();

### 2.2.2粘贴

QsciScintilla库提供的槽函数 virtual void QsciScintilla::paste();

### 2.2.3剪切

QsciScintilla库提供的槽函数 virtual void QsciScintilla::cut();

## 2.3窗口显示功能

### 2.3.1光标移动

首先利用函数库（conio.h \*非标准库）与主机交互，建立连接，获得一个句柄；设置起始位置，（这里是将输入区域看成一个XY轴的第一象限），获取屏幕缓冲区信息；通过不断受到机器传到缓冲区的信息，来识别传过来的信息内容并做出相对应的反应；根据键码两位16进制编码来区分信息内容，以此来更改光标的坐标。

### 2.3.2行号设置与跳转

行号显示通过Qsciscintilla外部库进行添加，进行设定行号显示位置，宽度，大小。行号跳转通过查询当前文本光标位置，在设置光标位置移动到指定位置即可。同时这顶光标所在位置高亮。

### 2.3.3关键字识别

定义一个heighlight类进行关键字录入，通过设定正则表达式来让对应的关键字进行不同颜色的显示。通过Qsciscintilla中的c语言表达来进行显示，另外还增加了一些其他的关键字信息。

### 2.3.4括号匹配与补全

括号匹配通过Qsciscintilla外部库进行添加。而括号补全则是通过继承QObject类对键盘事件进行监听。当判断用户输入的括号进行相应光标位置的对应的括号添加。

### 2.3.5自动排版

通过自定义一个类继承自QObject然后在对键盘进行监听，当判断输入是；时进行代码自动格式化，在通过对一行的字符串进行排版，主要通过=或者花括号等进行依次判断和添加。

### 2.3.6多行注释

首先获取用户选中区域的起始行号、末尾行号

editor->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

editor->*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

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

editor->*removeSelectedText*();

### 2.3.7注释隐藏/显示

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：位置 – 值：查找到的内容”的形式存入数组中，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。

更新注释：每当文本改变时，根据改变位置更改数组中每个元素的位置值。

显示注释：从vector中获取一个结构体，在当前文本编辑区内指定位置插入注释。循环此过程直至遍历vector中所结构体。

### 2.3.8代码块折叠

首先在编辑框的左侧边栏处添加折叠符号之后通过调用Qscintilla中的折叠函数来实现折叠操作editor->*setFolding*(QsciScintilla::BoxedTreeFoldStyle, 注：左侧边栏的标识符一共有32个，当实现注释显示或隐藏的时候会用到八个标识符，注意不要与注释折叠功能所用标识符发生冲突。

## 2.4文件资源管理功能

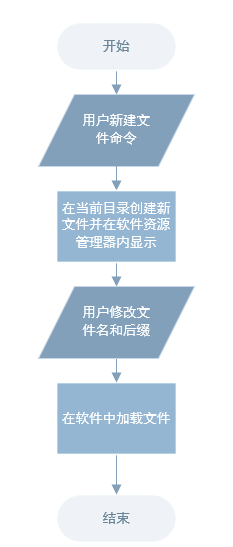
### 2.4.1新建文件

在用户点击新建时，在软件资源管理器下新建文件，并由用户修改名称和后缀

textEdit->*clear*();

setCurrentFileName("");

**流程图：**



### 2.4.2打开文件

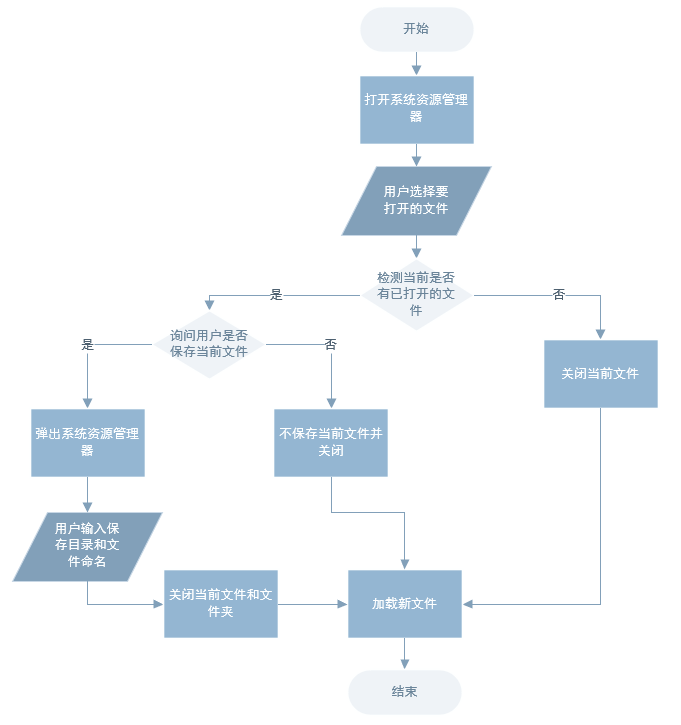
QTextStream File\_in(&file);

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

setCurrentFileName(fileName);

**流程图：**

**流程图：**

**流程图：**

### 2.4.3保存文件

通过调用QTextStream类库作为与Windows资源管理器的接口，用户将可以使用资源管理器自带的各类弹窗选择想要保存的路径，并填写文件名，再保存文件。

保存时需要通过判断语句判断是否为初次保存，如果初次保存则需要弹窗让用户选择保存路径，否则直接写入之前保存的文件。

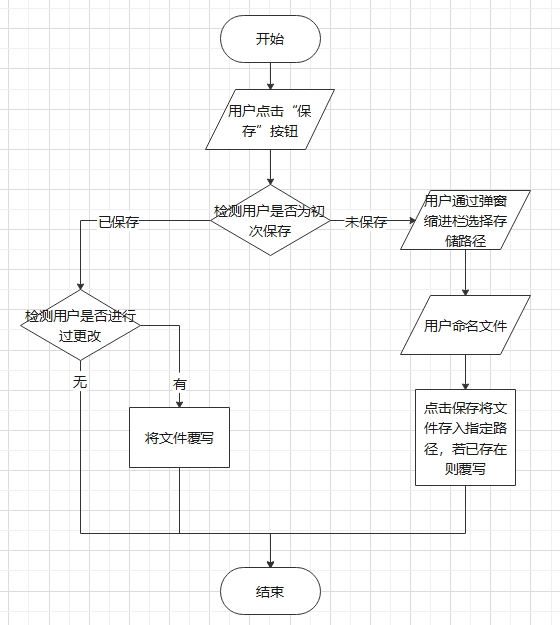
#include < QTextStream >

QString fileName = QFileDialog::getSaveFileName(this);

File\_out(&file);

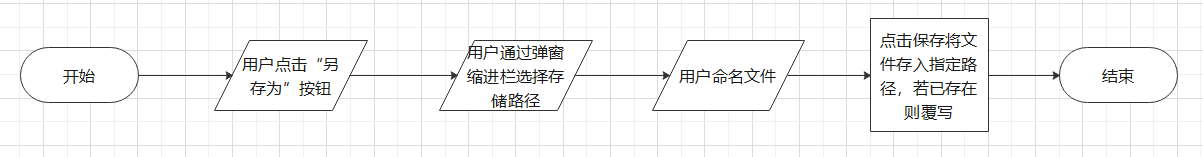
File\_out << textEdit->text();

**流程图：**



### 2.4.4另存文件

以与Windows资源管理器的接口QTextStream库为函数来源，用户将可以使用资源管理器自带的各类弹窗选择想要保存的路径，并填写文件名，再保存文件。保存时需要判断路径中是否存在重名文件，选择是否替换，此功能将由资源管理器解决。

**流程图：**

### 2.4.5文件资源管理区

在用户进行打开或新建文件的操作时，首先会通过QFileDialog::getOpenFileName()获取到文件的绝对路径。通过

QFileInfoList MainWindow::allFile递归循环遍历将该文件所在的文件夹下的所有文件名后缀为“.c”的绝对路径储存下来并且通过TreeWidgetItem依次储存下来并且添加到文件资源界面所在的TreeWidget上。通过信号和槽的连接机制使得对资源管理界面上的点击事件进行识别。同时需要判断所点击的节点是哪一类，之后反向相应的操作：文件跳转打开、函数跳转。

## 2.5调试编译运行功能

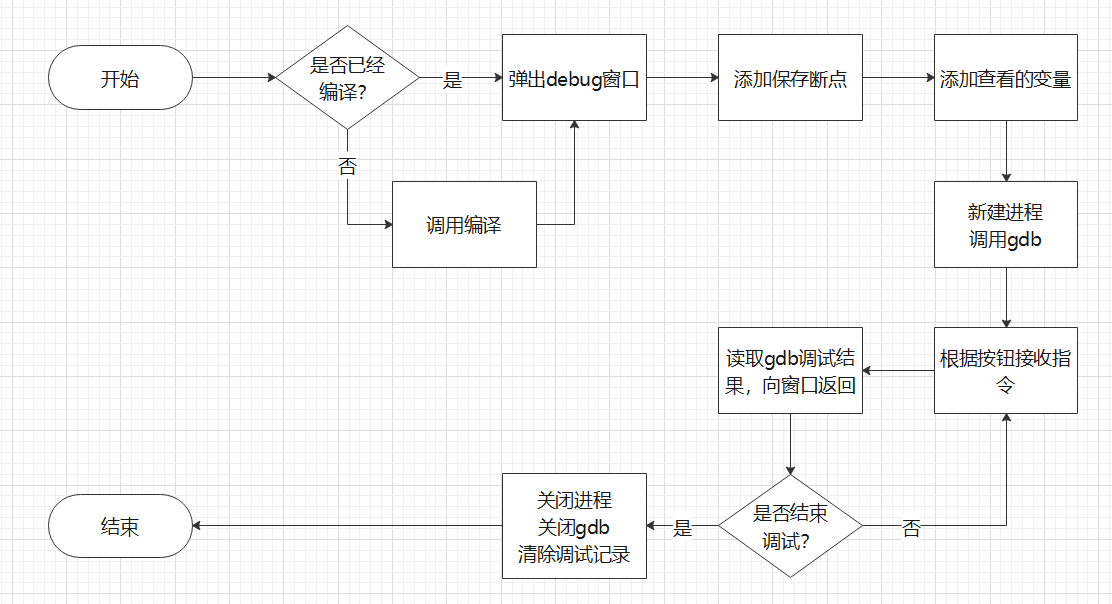
### 2.5.1调试功能

用户执行debug选项后，首先根据是否经过编译的判断来执行MainWindow::compCode()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和用户通过点击编辑区边栏设置的断点信息，并开启DebugThread线程，通过QProcess运行gdb开始调试。

调试过程中，程序通过DebugDialog接收用户操作指令，将其转换为gdb指令并通过QProcess::write()写入cmd传递给gdb程序。从gdb输出和错误文件中读取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。

**流程图：**



### 2.5.2编译功能

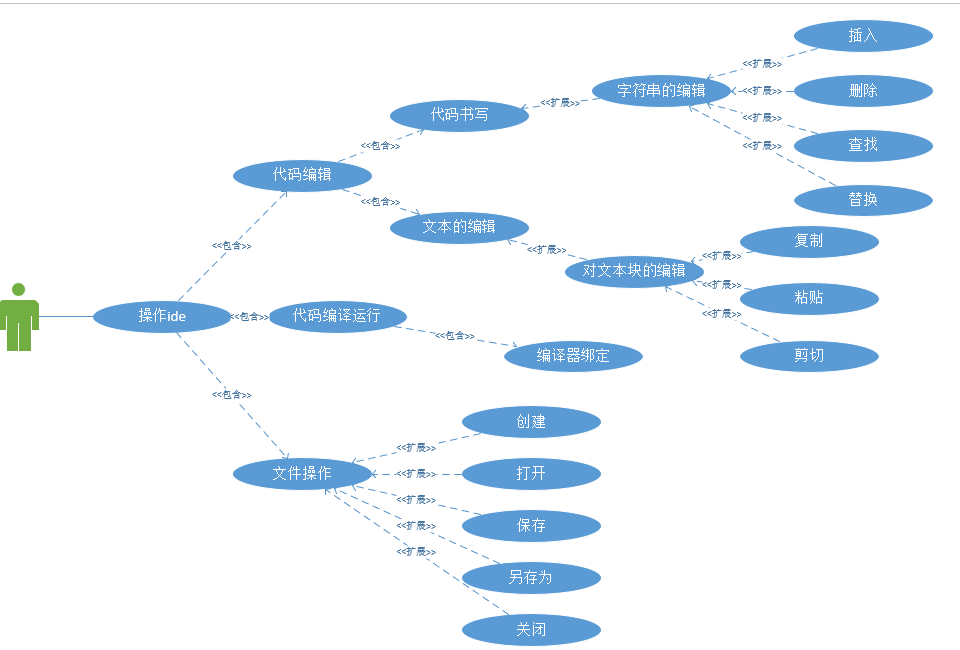
首先判断文件是否发生过改变，如果发生过则进行提示弹窗是否保存。之后进行预编译操作，生成在此基础之上的”.c”文件，为了接下来的debug做准备。之后进行编译，调用gcc将该文件生成在当前文件夹下的exe文件。同时通过QProcess::powerShell->readAllStandardOutput和QProcess::powerShell->readAllErrorOutput读取消cmd窗口消息。将消息进行回显。

### 2.5.3运行功能

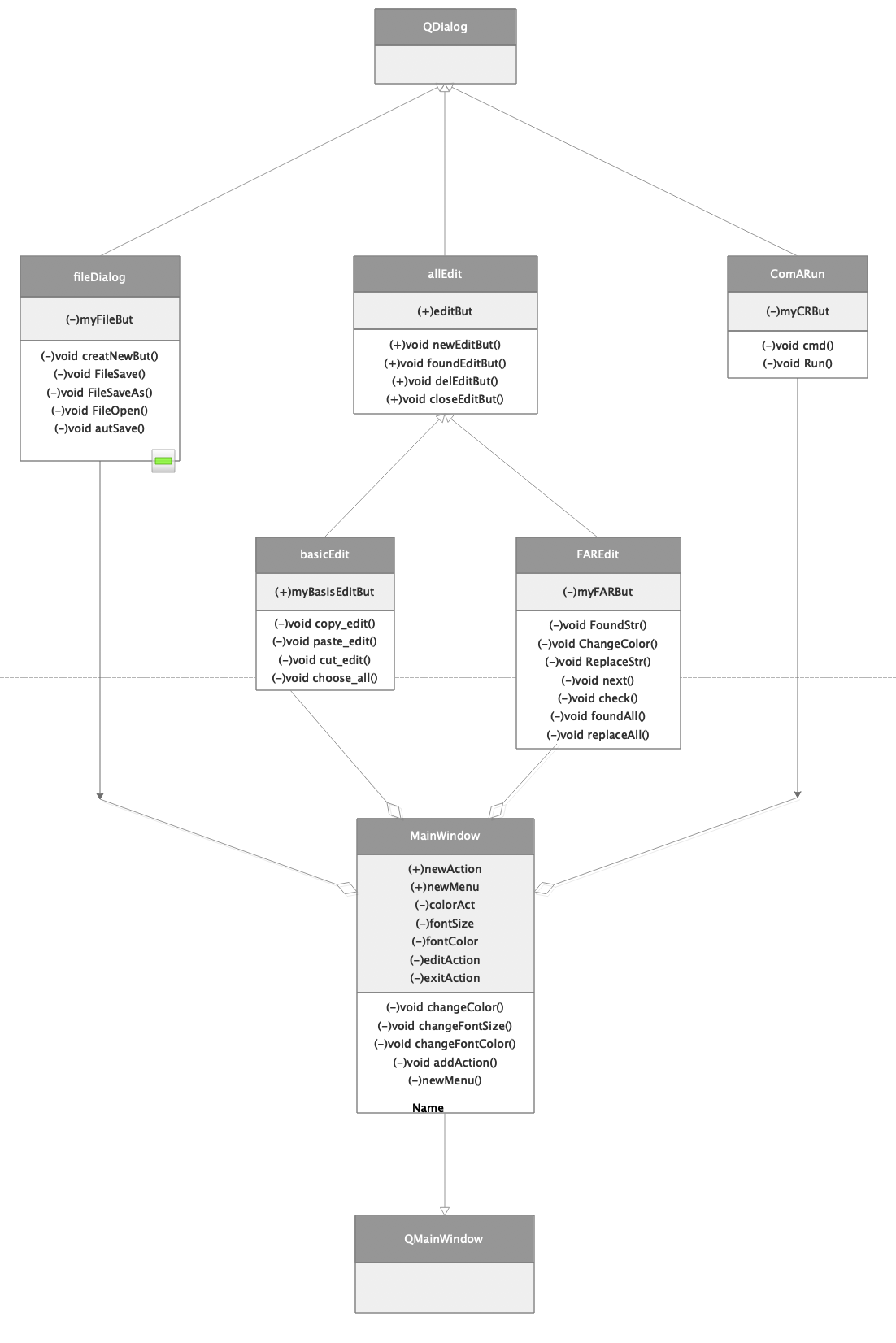
调用对之前gcc编译所生成的exe文件进行运行，弹出cmd窗口，并且进行操作和实时的输入输出。

# 3.UML图示分析

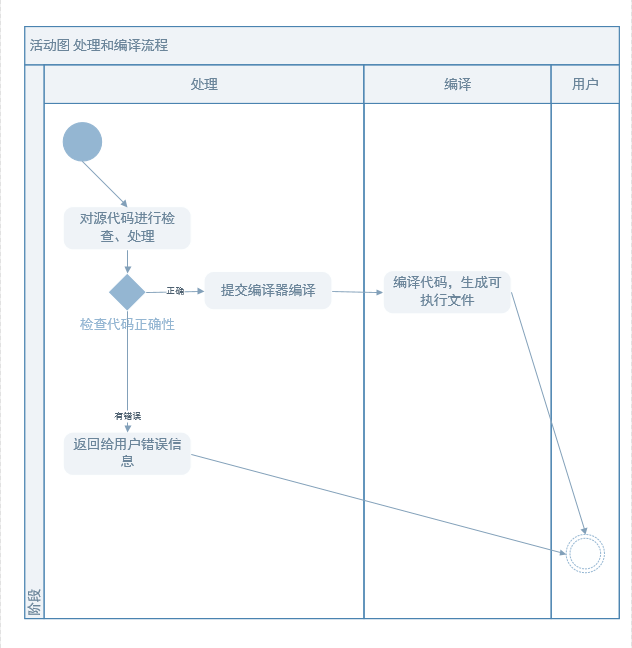
## 3.1用例图



## 3.2类图



## 3.3活动图



## 3.4状态图

