

浙江大学

Object-Oriented Programming 期末 project 报告



学生姓名: 蒋仕彪 学号: 3170102587

学生姓名: 刘明锐 学号: 3170105696

学生姓名: 刘书含 学号: 3170105283

学生姓名: 李广林 学号: 3170104648

学生姓名: 吕耀维 学号: 3171014185

2018~2019 春夏学期 2019 年 6 月

1. 需求分析

本组选择题目为文本编辑器，即用于文本和代码的及时编辑、修改、保存功能，小组成员结合选题文档要求与自己对于项目的想法，从 Sublime 和 Notepad++ 中汲取灵感，在中期报告中提出了本项目的基本、拓展和高级功能，并在实现过程中根据实际情况进行了一定的调整与功能上的增减，最终完成了一个文本编辑器。

我们将它命名为 SubCode。

2. 功能介绍

2.1 基本功能

1. 文本文件的新建、读取、保存、格式转换

- 新建文本

当用户需要新建文本文件的时候，只用点击 new 的图标或者轻敲快捷键 Ctrl+N，如果在当前文件修改且未保存的情况下，会提示用户进行保存并新建文件。可以新建后缀名为 .cpp, .txt, .py 等的文件。

- 打开文本

用户还能够从文件中读取文本。用户可以选择本地任意路径下的文本文件进行查看，只要该文件的后缀名是 .cpp, .txt, .py 之一。

- 保存文本

我们还支持通过点击 save 或者快捷键 Ctrl+S 来保存已编辑的文件内容；同时，如果选择了 save as，修改后缀名就可以直接完成文本格式的转换。在用户关闭窗口时，如果当前文件未保存，会有提示窗弹出，增加用户的友好使用体验。

2. 文本文件的编辑

- 插入和删除

项目会显示当前光标所在的位置，在此基础上可以实现字符的插入及删除。当用户输入内容时，光标处会自动插入用户所输入的中文字符、英文字符、特殊符号、数字等。可以通过移动鼠标并点击特定位置或者按键盘中的上下键来更改当前光标所在的位置。当用户输入出现错误时，可以按下键盘上的 delete（删除光标后字符）、Backspace（删除光标前字符）来实现删除。

- 文本选择

文本选定可以通过鼠标选定，或者按住 Shift 通过键盘方向键移动光标来进行文本选定，同时支持右键->select all 或者 Ctrl + A 进行全选。

- 重做和撤销

关于重做和撤销。当用户希望撤销当前编辑步骤时，只需要点击撤销图标或 Edit->Undo 或 Ctrl+Z 或右键->Undo 就能完成撤销；相应的，如果需要重做最近一次撤销的操作，只需要点击重做图标或 Edit->Redo 或者 Ctrl+Y 或者右键->Redo 就能完成重做。

- 剪切、复制和粘贴

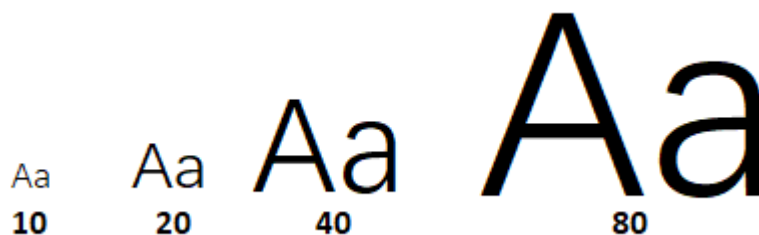
关于剪切、复制、粘贴。当用户需要对文本进行批量编辑时，一定会使用到剪切复制粘贴的功能。我们需要实现通过 Edit->Cut 或 Ctrl + X 或右键->Cut 能够剪切选中文本，通过 Edit->Copy 或 Ctrl + C 或右键->Copy 能够复制选中文本，通过 Edit->Paste 或 Ctrl + V 或右键->Paste 能够粘贴剪贴板内容到指定位置。

2.2 拓展功能

1. 文本文件的美化编辑

- 字号

具体地，字号支持从 1 磅到 120 磅的显示。可以输入数字直接对选择的文字进行调整，也可以点击调整按钮实时调整字体大小。



- 颜色

我们本来写完了颜色的功能，后来一致觉得，作为合格的文本编辑器，应该不支持用户对字体颜色的自定义调整，最后删掉了这个功能。

- 字体

我们需要实现的功能是，用户的电脑自带的字体全部可以在我们的文本编辑器中完整的使用。下面以现有程序作为案例说明。下图是完全能够正常显示的中文字体，当然肯定支持对西文字体的显示。

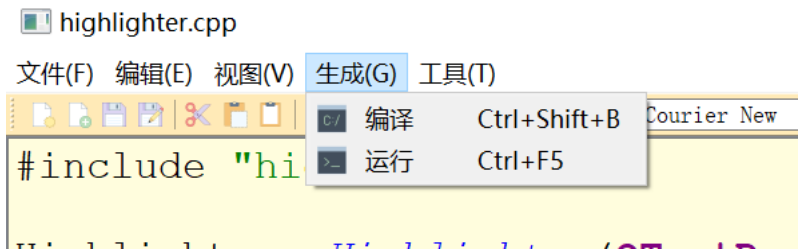
2.3 高级功能

作为 IDE 工具的编译功能

作为一个 IDE 工具，我们的文本编辑器要实现对代码文件快速编译运行的功能。通过读取文件名信息，识别文件后缀名确定文件的代码类型

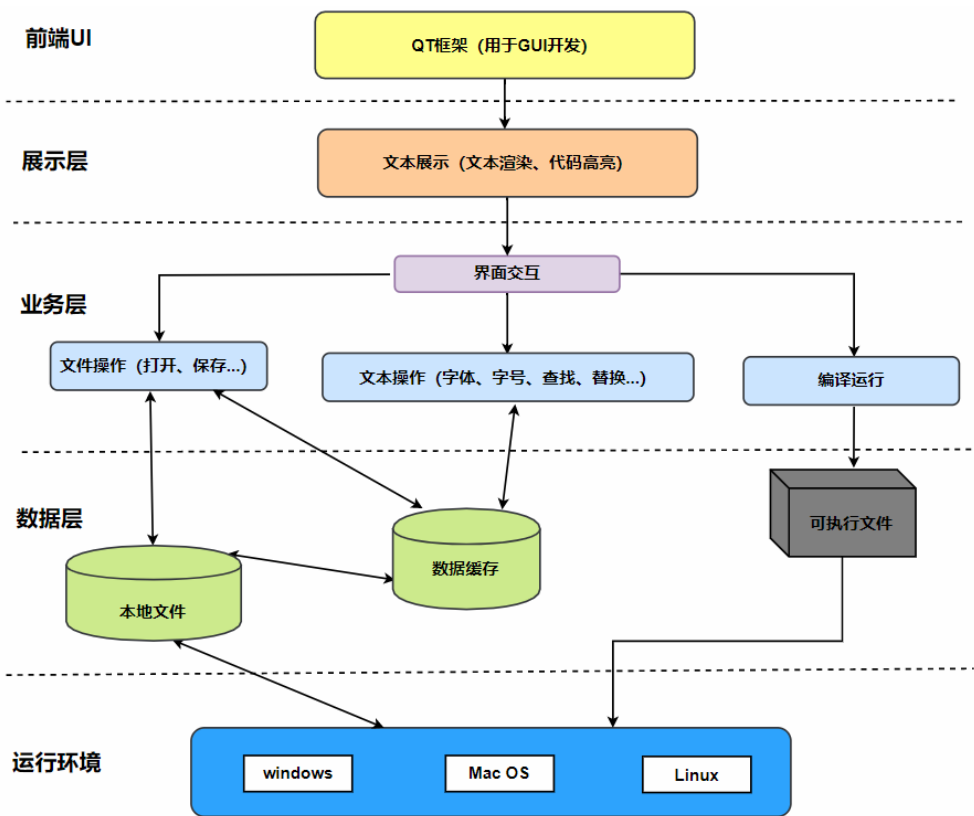
（如 .cpp .py ），再调用对应编译器的编译命令对文件进行编译运行。比如，对后缀名为 cpp 的文件使用 g++ 编译器，对后缀名为 py 的文件使用 python。

如果编译错误，我们编写的 IDE 项目还将返回具体的报错信息，并且提示哪一部分的代码出现了什么样的错误。在提示框的最后还会显示 warning 高亮以及完整的错误列表提示。



3. 关键设计思路与方法

3.1 该工程总架构图



3.2 前端 UI 与展示层

前端 UI 与展示层对应三层结构中的表示层，主要承担的功能是向用户展现软件的图形化界面、向用户展现当前处理图像的预览图、向用户提供友好的在软件界面上和在预览界面上的图形化交互方式。

前端 UI 主要通过 Qt 框架的界面库进行开发，利用 Qt Designer UI 设计工具与 QT 界面库高效完成前端界面设计。

界面按钮的交互和画布的交互主要利用 Qt 的信号槽机制来实现通过轮询检测时间的发生并调用对应的若干函数；利用信号槽，我们可以捕捉到界面上按键的触发，鼠标的点击、按下、移动、松开等操作，以及键盘组合键的触发，从而为用户提供多样化的交互方式并完成与业务逻辑层进行对接的功能。

3.3 业务逻辑层

业务逻辑层是项目的主体部分，从上部的表示层接收用户的操作指令，并对下部的数据层执行数据处理的业务，再将相应的预览图显示数据传回，其主要功能有：

- 执行文本框的复制、剪切、粘贴操作
- 实现缩放字号、选择字体、修改主题等操作
- 执行文本的搜索和查找功能
- 实现文本的保存和另存功能
- 编译和运行

所有操作都会与数据缓存交互。

特殊地，编译和运行会生成可执行文件；文件操作会对本地文件（硬盘）进行交互。

3.3 数据层

数据层的主要功能是对本地文件的读写以及相关的数据格式处理等功能；它向上层的业务层提供由文件得来的文本信息，也能将当前文本信息保存为相应的本地数据文件。

2.4 运行环境

得益于 Qt 框架的跨平台特性，我们可以以极小的代价将项目发布为可以在 Windows、Mac OS 或 Linux 系统上运行的应用。值得一提的是，小组成员同样也将不同的开发环境中参与到项目的开发中来。

4. 详细设计

为了 GUI 程序的开发便捷、降低整个项目的开发难度、提高程序的稳定性和兼容性，我们使用了 Qt 框架作为项目整体实现的总框架。

下面我们将分成 **UI 的约定和设计**、**文本编辑功能**、**保存机制和文件操作**、**编译和运行**、**语法高亮** 等几个方面来说明我们的设计思路和方法。

4.1 UI 的约定和设计

- 整体设计准则

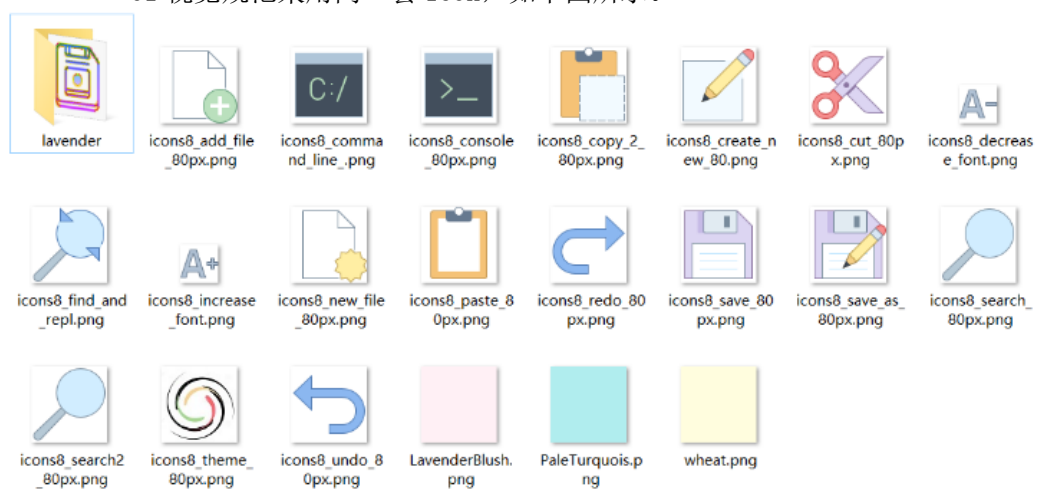
因为是 IDE，保持交互直接性较为重要，所以在整体的视觉遵照了简洁、直观的设计原则。初始默认主视觉为浅黄色主题，最上方是菜单栏，然后是工具栏，最后是编辑框。提供了三套配色方案，结合 QSS 和各个组件自己的设置函数控制 ui，最初用 Qt 自带的 designer 画出框架，细致的配色及高亮设置则使用 QSS 样式。通过更换 stylesheet 文件达到切换主题的目的。

Qss 语法和 Css 非常类似，有多种选择器描述方式，在本框架中只需要声明类名，然后设置关键字及其值。下面以 Menubar 为例，首先写出选择类，接着，如果有子内容/名字就紧跟标注，大括号中写入详细样式，这里描述的就是菜单栏中的独立菜单普通放置的格式。

```
QMenuBar::item {  
    spacing: 3px; /* spacing between menu bar items */  
    background: transparent;  
    color: #444342;  
    padding: 0px 10px;  
}
```

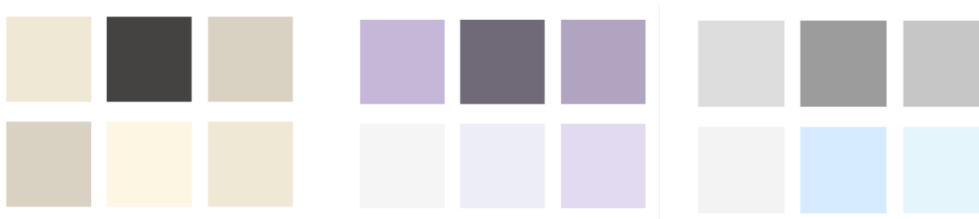
样式切换总的实现逻辑是：第一步，对各个主题配色方案的 QSS 文件完成设计；第二步，将 qss 文件存入资源文件夹并导入 qrc；第三步，连接主题设置按钮和 stylesheet 设置函数，当菜单栏主题 select 被触发时，调用 qss 设置函数设置相应主题。

UI 视觉规范采用同一套 icon，如下图所示。

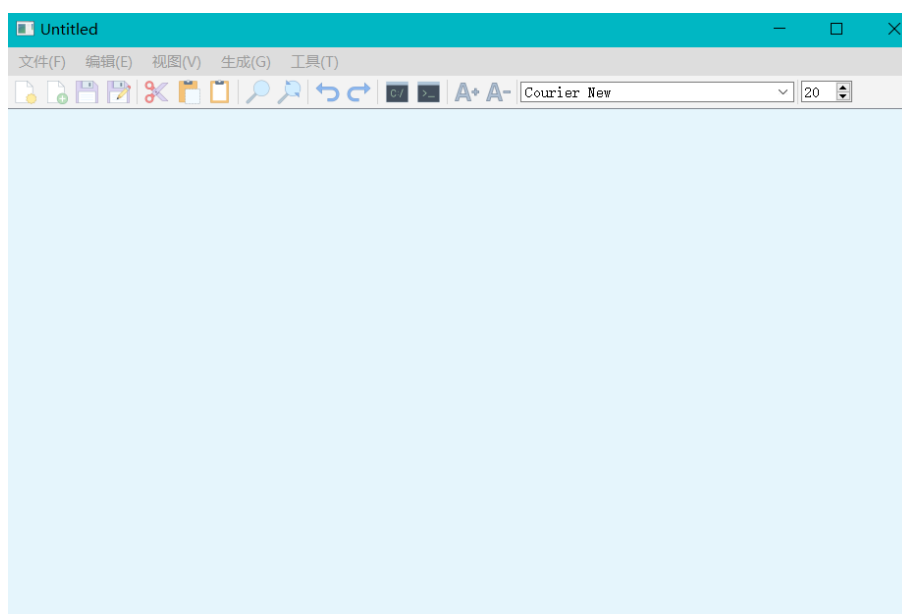
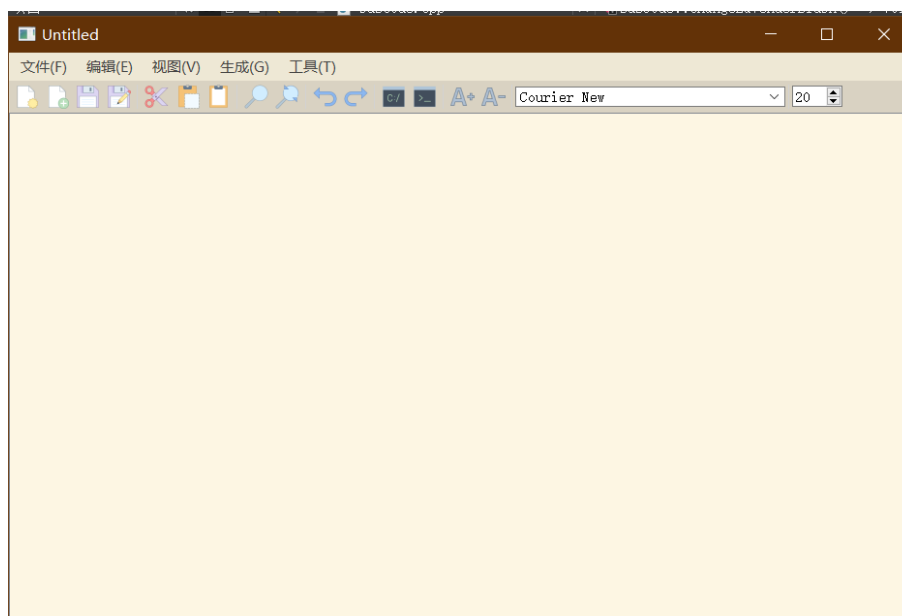


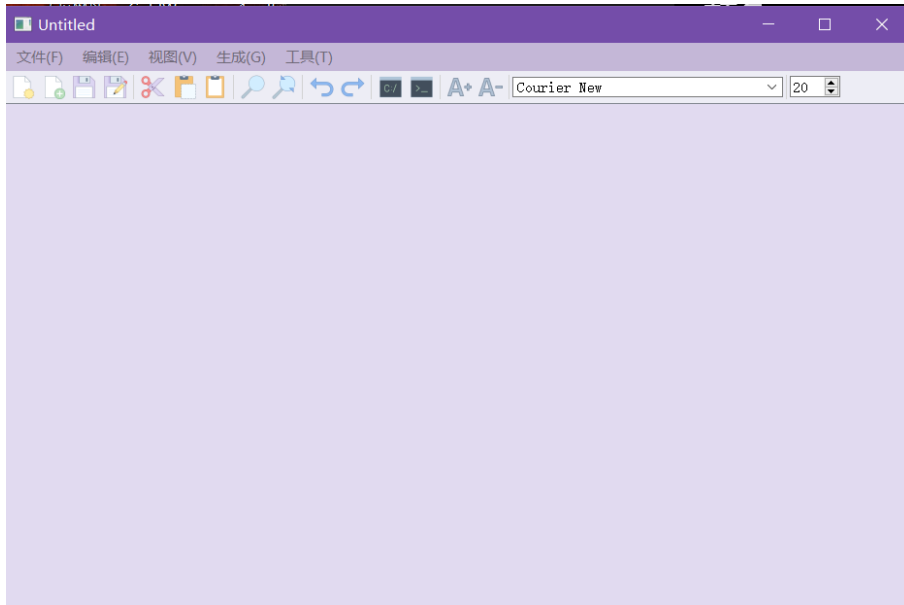
最终使用三套配色方案，分别是 wheat, lavender, Turquoise。

规范色卡依次展示如下：



界面预览依次展示如下：

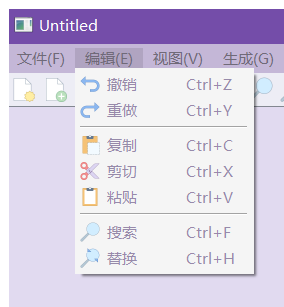




- 菜单栏

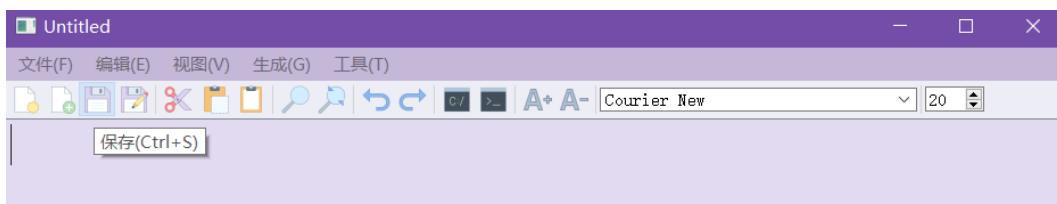
菜单栏分为文件、编辑、视图、生成、工具，参考了 VS 和 Sublime 等现存 IDE 的菜单栏需求设计，将文件级操作（新建、打开、保存、另存为）放入文件菜单、将基础编辑操作（剪切、粘贴、复制、撤销、重做、搜索、替换）放入编辑菜单栏、主题编辑视图操作（主题切换、总体字体调整）放入视图菜单栏、将程序编译执行工具（编译、运行）放入生成菜单栏。

下面以编辑菜单栏为例，将 action 放入新建菜单标注，给出图标和快捷键提示，并且根据子任务的实际意义再进行分类放置。



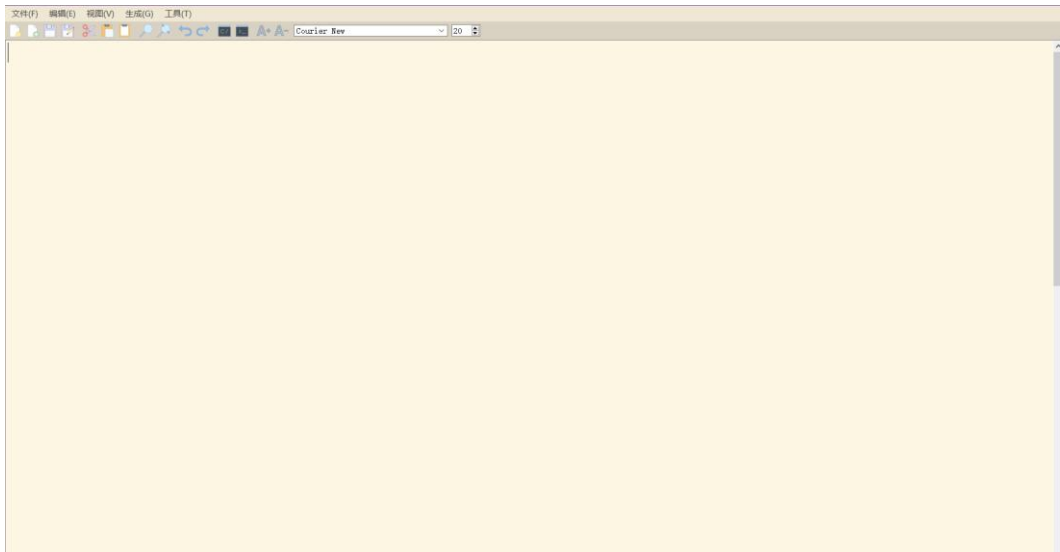
- 工具栏

工具栏放置首先按照功能分组，然后根据菜单栏的顺序依次从左到右展开。为了更好的交互性，主要实现的内容有：第一，给 action 增加快捷键即快捷键提示，当鼠标移到图标上时会有文字提示用户当前快捷键是如何调用的。第二，连接图标和实现的相应内容操作。第三，避免误操作将工具栏位置锁定，禁止拖动和悬浮。实现的方式是把 movable 状态一直设置为 false。



- 编辑框

调用 Qt 自带的 `TextEdit` 放置在主 widget 内，从 widget 继承样式。然后给 `textedit` 增加滚动条，当用户输入内容时自动换行，当超出当前框后，滚动条自己出现。



4.2 文本的基本功能

- 文本的剪切、复制和粘贴

文本的剪切、复制和粘贴是文本编辑器基础但重要的功能之一。

在 UI 设计中，将剪切、复制和粘贴依次放入 `mainToolBar` 以及菜单栏中，并配上相应的图片。



然后我们把键盘行为的 “`ctrl+X`”，“`ctrl+C`”，“`ctrl+V`” 与它们联系。

在 UI 约定里，剪切是 `actioncut`，复制是 `actioncopy`，粘贴是 `actionpaste`。所以在 `Subcode` 的构造函数里，我们还要声明它们与 `Action` 的触发机制。

```
connect(ui->actionpaste,    SIGNAL(triggered(bool)), ui->textEdit, SLOT(paste()));
connect(ui->actioncopy,     SIGNAL(triggered(bool)), ui->textEdit, SLOT(copy()));
connect(ui->actioncut,      SIGNAL(triggered(bool)), ui->textEdit, SLOT(cut()));
```

- 文本的撤销和重做

文件的撤销和重做也很基础、重要。

在 UI 设计中，将剪切、复制和粘贴依次放入 `mainToolBar` 以及菜单栏中，并配上相应的图片。



然后我们把键盘行为的 “ctrl+X”，“ctrl+C”，“ctrl+V” 与它们联系。

在 UI 约定里，撤销是 `actionundo`，重做是 `actionredo`，所以在 `Subcode` 的构造函数里，我们还要声明它们与 `Action` 的触发机制。

```
connect(ui->actionundo, SIGNAL(triggered(bool)), ui->textEdit, SLOT(undo()));  
connect(ui->actionredo, SIGNAL(triggered(bool)), ui->textEdit, SLOT(redo()));
```

4.3 文本的字号、字体、主题功能

- 字号和字体

因为是模仿 sublime 功能的文本编辑器，我们并不支持“富文本编辑器”，而是规定字号必须整体修改（其实部分修改原理也一样）。

QT 里相关函数有点多，而且不同类里面都或多或少有字号和字体的一些函数操作（但有些使用起来是假的），所以如何甄选和使用它们就比较复杂了。

我们的主要思想是：先在 `Subcode` 建立时，创建一个 `QFontComboBox`（用来选择字体）和 `QSpinBox`（用来选择字号）。在之后的字号变大变小、字体变化时，这两个对象也跟着动，然后字体和字号实时与它们对齐。

主程序开始时我们要加上这样几行来初始化：

```
fontComboBox = new QFontComboBox(this);  
fontSizeSpinBox = new QSpinBox(this);  
ui->textEdit->setFont(QFont("Courier New", 20));  
fontSizeSpinBox->setRange(1, 200);  
fontSizeSpinBox->setValue(20);  
fontComboBox->setCurrentFont(QFont("Courier New", 20));  
ui->mainToolBar->addWidget(fontComboBox);  
ui->mainToolBar->addSeparator();  
ui->mainToolBar->addWidget(fontSizeSpinBox);
```

然后我们要设置一些触发的机制。

除了字号和字体的显示选择，注意还有快捷变化字号：



所以我们给出对应的 `connect` 函数。

```
connect(ui->actionsmaller, SIGNAL(triggered()), this, SLOT(Smaller()));  
connect(ui->actionlarger, SIGNAL(triggered()), this, SLOT(Bigger()));
```

```
connect(ui->textEdit, SIGNAL(textChanged()), this, SLOT(fileChange()));
connect(fontSizeSpinBox, SIGNAL(valueChanged(int)), this, SLOT(fontChange()));
connect(fontComboBox, SIGNAL(currentFontChanged(const QFont &)), this,
SLOT(fontChange()));
```

这里，我们将字号和字体的变化视为同一种变化，并建立相同的触发函数。

```
void SubCode::fontChange() {
    QFont now = fontComboBox->currentFont();
    now.setPointSize(fontSizeSpinBox->value());
    ui->textEdit->setFont(now);
}

void SubCode::Smaller() {
    int size = fontSizeSpinBox->value() - 1;
    if (size < 1) size = 1;
    fontSizeSpinBox->setValue(size);
    fontChange();
}

void SubCode::Bigger() {
    int size = fontSizeSpinBox->value() + 1;
    if (size > 120) size = 120;
    fontSizeSpinBox->setValue(size);
    fontChange();
}
```

● 主题的切换

主题只有原来的黄色是不是有点单调？

所以我们要解决背景变化的问题。

QT 的很多类都有 **palette** 这个变量，用来保存各种颜色信息。用户想要切换主题时，我们只需对 `ui->textedit` 进行背景颜色的调整，其中参数是 **QPalette::Base**。拿其中的一些分格切换来举例。

建立各种主题和函数之间的 connect 联系后，我们直接去设置各种变换函数。

```
void SubCode::changeWheat() {
    QPalette palette = ui->textEdit->palette();
    palette.setColor(QPalette::Base, QColor(255, 253, 221));
    ui->textEdit->setPalette(palette);
}

void SubCode::changePaleTurquoise() {
```

```
    QPalette palette = ui->textEdit->palette();
    palette.setColor(QPalette::Base, QColor(175, 238, 238));
    ui->textEdit->setPalette(palette);
}

void SubCode::changeLavenderBlush() {
    QPalette palette = ui->textEdit->palette();
    palette.setColor(QPalette::Base, QColor(255, 240, 245));
    ui->textEdit->setPalette(palette);
}
```

4.4 文本的搜索、替换功能

文本的搜索和替换看似很简单，其实逻辑蛮复杂的。

为了防止代码过于臃肿，我们新开一个 `search.h` 和 `search.cpp`，专门用于搜索和替换。既然如此，我们就可以重新定义一个搜索类了。由于搜索框是 `QDialog`，很自然地就会从 `QDialog` 那儿继承来一个类。

```
class Search :public QDialog
{
    Q_OBJECT
public:
    Search(bool isreplace, QWidget *parent = nullptr);
    bool searching;

private:
    Ui::Search *ui;

signals:
    void findNext(const QString &, Qt::CaseSensitivity, bool);
    void replaceNext(const QString &, const QString &, Qt::CaseSensitivity);
    void replaceAll (const QString &, const QString &, Qt::CaseSensitivity);
    void searchModeChange(int);

private slots:
    void on_buttonBox_accepted();
    void catchModeChange(int);
};
```

为了防止代码臃肿，与用户交互的按钮和界面，可以直接在 `search.ui` 里实现。



为此要新建一些 QLineEdit, Qcheckbox 等部件, 并在 search.cpp 里接收。我们根据用户的搜索/替换需求, 发射不同的信号给主程序。

```
void Search::on_buttonBox_accepted() {

    int type = ui->tabWidget->currentIndex();
    if (type == 0) {
        //查找功能
        if (ui->lineEdit1->text().isEmpty()) {
            QMessageBox::warning(this, QString::fromLocal8Bit("warning"),
            QString::fromLocal8Bit("Searching text can't be empty!"), QMessageBox::Yes);
            this->show(); return;
        }
        Qt::CaseSensitivity Case = ui->checkBox1->isChecked() ?
Qt::CaseSensitive : Qt::CaseInsensitive;
        emit findNext(ui->lineEdit1->text(), Case, ui->checkBox2->isChecked());
    }
    else {
        //替换功能
        qDebug() << "Start replace";
        if (ui->lineEdit2->text().isEmpty()) {
            QMessageBox::warning(this, tr("warning"), tr("Searching text can't be
empty!"), QMessageBox::Yes);
            this->show(); return;
        }
        if (ui->lineEdit3->text().isEmpty()) {
            QMessageBox::warning(this, tr("warning"), tr("Replacing text can't be
empty!"), QMessageBox::Yes);
            this->show(); return;
        }
        Qt::CaseSensitivity Case = ui->checkBox3->isChecked() ?
Qt::CaseSensitive : Qt::CaseInsensitive;
        if (ui->checkBox4->isChecked())
            emit replaceAll (ui->lineEdit2->text(), ui->lineEdit3->text(), Case);
        else
            emit replaceNext(ui->lineEdit2->text(), ui->lineEdit3->text(), Case);
    }
}
```

在主程序 subcode.cpp 里, 逻辑也较为复杂。我们拿“替换”举个例子。

当主程序刚收到替换请求时, 立即做出一些 connect 的连接操作:

```

void SubCode::startReplace() {
    qDebug() << "Start replace2";
    search = new Search(true);
    search->show();
    connect(search, SIGNAL(searchModeChange(int)), this,
    SLOT(searchModeChange(int)));
    connect(search, SIGNAL(replaceNext(const QString &, const QString &,
    Qt::CaseSensitivity)), this, SLOT(replaceNext(const QString &, const QString &,
    Qt::CaseSensitivity)));
    connect(search, SIGNAL(replaceAll(const QString &, const QString &,
    Qt::CaseSensitivity)), this, SLOT(replaceAll(const QString &, const QString &,
    Qt::CaseSensitivity)));
}

```

这些 connect 起到了 search.cpp 和 subcode.cpp 里的函数的交接。

- 考虑写一个“替换下一个”的操作：
 - 替换的第一步是查询，我们用 textEdit 自带的查询函数去查询，默认就是以光标当前位置的，注意还要标注大小写是否敏感。
 - 如果查询不到，返回错误信息。
 - 此时肯定是呈选中状态的，我们直接用 textCursor().deleteChar() 来删除。
 - 此时光标位置依然是正确的，我们用 textCursor().insertText(str2) 来插入。

```

bool SubCode::replaceNext(const QString &str1, const QString &str2,
    Qt::CaseSensitivity cs)
{
    bool findSucc = cs == Qt::CaseInsensitive ? ui->textEdit->find(str1) :
    ui->textEdit->find(str1, QTextDocument::FindCaseSensitively);
    if (!findSucc)
    {
        QMessageBox::information(this, tr("SubCode"), tr("Can't find
        \"%1\".").arg(str1), QMessageBox::StandardButton::Ok);
        search->show();
        return false;
    }
    ui->textEdit->textCursor().deleteChar();
    ui->textEdit->textCursor().insertText(str2);
    search->show();
    return true;
}

```


- 考虑写一个“替换全部”的操作：
 - 查询前，我们先要用 `moveCursor` 将鼠标移动到开头。
 - 接着外层是一个大循环：每次执行一次查询，并直接执行替换操作。
 - 如果结束后替换数目是 0，返回错误信息。
 - 否则返回替换的数量。

```
bool SubCode::replaceAll(const QString &str1, const QString &str2,
Qt::CaseSensitivity cs)
{
    ui->textEdit->moveCursor(QTextCursor::Start);
    int replacenum = 0;
    while (true){
        bool findSucc = cs == Qt::CaseInsensitive ? ui->textEdit->find(str1)
                                                    : ui->textEdit->find(str1,
Qt::TextDocument::FindCaseSensitively);
        if (!findSucc) break;
        replacenum++;
        ui->textEdit->textCursor().deleteChar();
        ui->textEdit->textCursor().insertText(str2);
    }
    if (!replacenum)
    {
        QMessageBox::information(this, tr("SubCode"), tr("Can't find
\\\"%1\\\".").arg(str1), QMessageBox::StandardButton::Ok);
        search->show(); return false;
    }
    QMessageBox::information(this, tr("SubCode"), tr("%1 texts have been
replaced.").arg(replacenum), QMessageBox::StandardButton::Ok);
    search->show(); return true;
}
```

4.5 保存机制和文件操作

文件保存机制也很繁琐，下面来详细介绍一下。

首先回忆一下，在另存为或者打开文档的时候，我们都要检查当前文件是否保存（如果没有，就要驳回那个操作）。这是一个重要的判断函数：

```
bool SubCode::checkSave() {
    if (isWindowModified())
    {
```

```

        int getMessage = QMessageBox::warning(this, tr("Subcode"),
        tr("The file has modified, do you want to save it?\n"),
        QMessageBox::Yes | QMessageBox::No | QMessageBox::Cancel);
        if (getMessage == QMessageBox::Yes) return saveFile();
        if (getMessage == QMessageBox::Cancel) return false;
    }
    return true;
}

```

注意到它是有一个返回值的。如果返回 false 的话，相当于当前操作被驳回。

还有一个比较重要的函数是：**setStatus**。它是用来显示在编辑器最上面的文件名。注意到会传入参数 modified，表示目前文本状态（1：已经编辑过 0：成功保存）。我们同时实时维护 WindowModified()，这样如果我们在 setWindowTitle 的字符串后加上 “[*]”，系统会自动根据 WindowModified 的值来判断是否加入 *。

```

void SubCode::setStatus(bool modified){
    QString title = curName.isEmpty() ? "Untitled" : QFileInfo(curName).fileName();
    title = title + "[*]";
    setWindowModified(modified);
    setWindowTitle(title);
}

```

下面介绍一下用来真正保存数据至文件的函数 **savetoFile**。流程很自然，我们打开目标文件名，将目前的 textEdit 转换为纯文本写进去（注意特判回车符号）。

写入完成后，注意要重置一下鼠标，并将状态清成 0。

```

bool SubCode::savetoFile(QString fileName){
    QFile file(fileName);
    file.open(QIODevice::WriteOnly | QIODevice::Text);
    QTextStream out(&file);
    QApplication::setOverrideCursor(Qt::WaitCursor);
    QString content = ui->textEdit->toPlainText();
    for (auto itr = content.cbegin(); itr < content.cend(); itr++){
        if (*itr == '\n') out << endl; else out << *itr;
    }
    QApplication::restoreOverrideCursor();
    Highlight(curName = fileName);
    setStatus(0); waiting = true;
    return true;
}

```

有了上述的铺垫，新建文件、打开文件、保存文件、另存为文件的逻辑会变得很清晰。我们只要根据不同的需求，调用相应的函数即可。

注意还有高亮的操作（后面会介绍），在新建文件的时候，默认以 txt 的格式进行高亮(相当于没有高亮)，名称也会标成 “Untitled”，这也是一般文本编辑器的做法。

```
bool SubCode::newFile() {
    if (!checkSave()) return false;
    curName = QString();
    ui->textEdit->clear();
    Highlight("1.txt");
    setStatus(0);
    return true;
}

bool SubCode::addFile() {
    if (!checkSave()) return false;
    bool result = false;
    QString fileName = QFileDialog::getOpenFileName(this, tr("Open text"), ".",
tr(FILETYPE));
    if (!fileName.isEmpty())
        result = openFile(fileName);
    return false;
}

bool SubCode::saveAs() {
    QString fileName = QFileDialog::getSaveFileName(this, tr("Save text"), ".",
tr(FILETYPE));
    if (!fileName.isEmpty())
        return savetoFile(fileName);
    return false;
}

bool SubCode::saveFile() {
    bool result = curName.isEmpty() ? saveAs() : savetoFile(curName);
    setStatus(0);
    return result;
}
```

4.6 编译和运行

编译和运行的逻辑比较清晰：

- 先判断是否保存，如果没有就驳回
- 根据后缀名，按 C++ 或 python 分类，如果都不是就报错。
- 根据文件名和所在路径，调用 cmd 命令来实现编译和运行。

```
bool SubCode::compile() {
    if (!checkSave()) {
        if (!saveFile()) return false;
    }
    QFileInfo file(curName);
    if (file.suffix() == "cpp") {
        QString cmd = "g++ " + file.filePath() + " -o " + file.baseName() + ".exe";
        qDebug() << cmd;
        if (system(cmd.toLatin1().data())) {
            QMessageBox::critical(this, tr("Subcode"), tr("Compiling error!"));
            return false;
        }
        return true;
    }
    if (file.suffix() == "py") {
        QString cmd = "python " + file.filePath();
        if (system(cmd.toLatin1().data())) {
            QMessageBox::critical(this, tr("Subcode"), tr("Compiling error!"));
            return false;
        }
        return true;
    }
    QMessageBox::critical(this, tr("Subcode"), tr("Only support C++ or Python"));
    return false;
}

bool SubCode::run() {
    if (!compile()) return false;
    QFileInfo file(curName);
    if (file.suffix() == "cpp") {
        QString cmd = file.baseName() + ".exe";
        if (system(cmd.toLatin1().data())) {
            QMessageBox::critical(this, tr("Subcode"), tr("Running error!"));
            return false;
        }
    }
    return true;
}
```



```

Highlighter::Highlighter(QTextDocument *parent, QString extensionName)
    : QSyntaxHighlighter(parent), highlightType(extensionName)
{
    if (highlightType == "cpp") /*c++代码高亮*/
    {
        HighlightingRule rule;

        /*长注释 特殊处理*/
        multiLineCommentFormat.setForeground(Qt::red);
        commentStartExpression = QRegularExpression("/\\*");
        commentEndExpression = QRegularExpression("\\*/");

        /*函数*/
        functionFormat.setFontItalic(true);
        functionFormat.setForeground(Qt::blue);
        rule.pattern = QRegularExpression("\\b[A-Za-z0-9_]+(?:\\()");
        rule.format = functionFormat;
        highlightingRules.append(rule);

        /*保留字*/
        keywordFormat.setForeground(Qt::darkBlue);
        keywordFormat.setFontWeight(QFont::Bold);
        QStringList keywordPatterns;
        keywordPatterns << "\\basmb" << "\\bauto" << "\\bbool" <<
            << "\\bbreak" << "\\bcase" << "\\bcatch" <<
            << "\\bchar" << "\\bclass" << "\\bconst" <<
            << "\\bconst_cast" << "\\bcontinue" <<
            << "\\bdefault" << "\\bdelete" << "\\bdo" <<
            << "\\bdouble" << "\\bdynamic_cast" <<
            << "\\belse" << "\\benum" << "\\bexplicit" <<
            << "\\bexport" << "\\bextern" << "\\bfalse" <<
            << "\\bfloat" << "\\bfor" <<
            << "\\bfriend" << "\\bgoto" << "\\bif" <<
            << "\\binline" << "\\bint" <<
            << "\\blong" << "\\bmutable" <<
            << "\\bnamespace" << "\\bnew" << "\\boperator" <<
            << "\\bprivate" << "\\bprotected" <<
            << "\\bpublic" << "\\bregister" << "\\breinterpret_cast" <<
            << "\\breturn" << "\\bshort" << "\\bsigned" <<
            << "\\bsizeof" << "\\bstatic" << "\\bstatic_cast" <<
            << "\\bstruct" << "\\bswitch" <<
            << "\\btemplate" << "\\bthis" <<
            << "\\bsignals" << "\\bsigned" <<

```

```

        << "\\bslots\\b" << "\\bstatic\\b" << "\\bstruct\\b"
        << "\\btemplate\\b" << "\\btypedef\\b" <<
"\\btypename\\b"
        << "\\bunion\\b" << "\\bunsigned\\b" <<
"\\bvirtual\\b"
        << "\\bvoid\\b" << "\\bvolatile\\b" << "\\bbool\\b";
foreach(const QString &pattern, keywordPatterns) {
    rule.pattern = QRegularExpression(pattern);
    rule.format = keywordFormat;
    highlightingRules.append(rule);
}

/*Q 类型*/
classFormat.setFontWeight(QFont::Bold);
classFormat.setForeground(Qt::darkMagenta);
rule.pattern = QRegularExpression("\\bQ[A-Za-z]+\\b");
rule.format = classFormat;
highlightingRules.append(rule);

/*常量字符串*/
quotationFormat.setForeground(Qt::darkGreen);
rule.pattern = QRegularExpression("\\\".*\\\"");
rule.format = quotationFormat;
highlightingRules.append(rule);

/*单行注释*/
singleLineCommentFormat.setForeground(Qt::red);
rule.pattern = QRegularExpression("//[^\n]*");
rule.format = singleLineCommentFormat;
highlightingRules.append(rule);
}
else if (highLightType == "py") /*python 代码高亮*/
{
    HighlightingRule rule;

    /*长注释 特殊处理*/
    multiLineCommentFormat.setForeground(Qt::red);
    commentStartExpression = QRegularExpression("'''");
    commentEndExpression = QRegularExpression("'''");

    /*函数*/
    functionFormat.setFontItalic(true);
    functionFormat.setForeground(Qt::blue);
    rule.pattern = QRegularExpression("\\b\\w+(?=\\()");

```

```

rule.format = functionFormat;
highlightingRules.append(rule);

/*保留字*/
keywordFormat.setForeground(Qt::darkBlue);
keywordFormat.setFontWeight(QFont::Bold);
QStringList keywordPatterns;
keywordPatterns << "\\band\\b" << "\\bas\\b" << "\\bassert\\b"
    << "\\bbreak\\b" << "\\bclass\\b" << "\\bcontinue\\b"
    << "\\bdef\\b" << "\\belif\\b" << "\\belse\\b"
    << "\\bexport\\b" << "\\bfinally\\b" << "\\bfor\\b"
    << "\\bif\\b" << "\\bimport\\b" << "\\bin\\b"
    << "\\bis\\b" << "\\blambda\\b" << "\\bnot\\b"
    << "\\bor\\b" << "\\bpass\\b" << "\\braise\\b"
    << "\\breturn\\b" << "\\btry\\b" << "\\bwhile\\b"
    << "\\bwhile\\b" << "\\bwith\\b" << "\\byield\\b"
    << "\\bTrue\\b" << "\\bFalse\\b" << "\\bNone\\b";
foreach(const QString &pattern, keywordPatterns) {
    rule.pattern = QRegularExpression(pattern);
    rule.format = keywordFormat;
    highlightingRules.append(rule);
}

/*常量字符串*/
quotationFormat.setForeground(Qt::darkGreen);
rule.pattern = QRegularExpression("\\\".*\\\"");
rule.format = quotationFormat;
highlightingRules.append(rule);
quotationFormat.setForeground(Qt::darkGreen);
rule.pattern = QRegularExpression("\\'.*'");
rule.format = quotationFormat;
highlightingRules.append(rule);

/*单行注释*/
singleLineCommentFormat.setForeground(Qt::red);
rule.pattern = QRegularExpression("#[^\n]*");
rule.format = singleLineCommentFormat;
highlightingRules.append(rule);
}
else /*无高亮*/
{
}
}

```


随后我们需要用 `highlightBlock` 修改文本格式产生高亮。大部分高亮可以通过 `QRegularExpressionMatchIterator` 查找符合正则表达式的位置，然后 `setFormat` 完成高亮，但是由于产生高亮以行为单位，多行注释需要特殊处理：

- 若目前该行刚刚开始，且上一行的高亮未结束，那么开头是一个高亮开始位置，否则找到第一个高亮开始位置。若不存在，结束该行高亮。
- 找到其之后第一个高亮结束位置，注意不能与高亮开始位置有任何重叠。如不存在，行末尾是一个高亮结束位置，注明高亮未结束。
- 高亮开始和结束之间的部分，跳到第一步。

```
void Highlighter::highlightBlock(const QString &text)
{
    if (highlightType == "cpp") /*c++代码高亮*/
    {
        /*标准高亮*/
        foreach(const HighlightingRule &rule, highlightingRules) {
            QRegularExpressionMatchIterator matchIterator =
rule.pattern.globalMatch(text);
            while (matchIterator.hasNext()) {
                QRegularExpressionMatch match =
matchIterator.next();
                setFormat(match.capturedStart(),
match.capturedLength(), rule.format);
            }
        }
        /*长注释 特殊处理*/
        setCurrentBlockState(0);

        int startIndex = 0;
        if (previousBlockState() != 1)
            startIndex = text.indexOf(commentStartExpression);

        while (startIndex >= 0) {
            QRegularExpressionMatch match =
commentEndExpression.match(text, startIndex);
            int endIndex = match.capturedStart();
            int commentLength = 0;
            if (endIndex == -1) {
                setCurrentBlockState(1);
                commentLength = text.length() - startIndex;
            }
            else {
                commentLength = endIndex - startIndex

```

```

+ match.capturedLength();
    }
    setFormat(startIndex, commentLength,
multilineCommentFormat);
    startIndex = text.indexOf(commentStartExpression,
startIndex + commentLength);
    }
}
else if (highLightType == "py") /*python 代码高亮*/
{
    /*标准高亮*/
    foreach(const HighlightingRule &rule, highlightingRules) {
        QRegularExpressionMatchIterator matchIterator =
rule.pattern.globalMatch(text);
        while (matchIterator.hasNext()) {
            QRegularExpressionMatch match =
matchIterator.next();
            setFormat(match.capturedStart(),
match.capturedLength(), rule.format);
        }
    }
    /*长注释 特殊处理*/
    setCurrentBlockState(0);

    int startIndex = -2;/**/
    if (previousBlockState() != 1)
        startIndex = text.indexOf(commentStartExpression);
    while (startIndex != -1) {
        QRegularExpressionMatch match;
        if (startIndex == -2) /*注意不能匹配到自身*/
        {
            startIndex = 0;
            match = commentEndExpression.match(text,
startIndex);
        }
        else match = commentEndExpression.match(text,
startIndex + 3);

        int endIndex = match.capturedStart();
        int commentLength = 0;
        if (endIndex == -1) {
            setCurrentBlockState(1);
            commentLength = text.length() - startIndex;
        }
    }
}

```

```

        else {
            commentLength = endIndex - startIndex
                + match.capturedLength();
        }
        setFormat(startIndex, commentLength,
multilineCommentFormat);
        startIndex = text.indexOf(commentStartExpression,
startIndex + commentLength);
    }
}
else /*无高亮*/
{
}
}

```

最后，由于 Highlighter 中高亮的具体格式在构造函数之后不能修改，因此每次改变高亮类型需要重新生成。

```

void SubCode::Highlight(QString curName) {
    QFileInfo sub(curName);
    delete highlighter;
    highlighter = new Highlighter(ui->textEdit->document(), sub.suffix());
}

Highlight(curName = fileName);

```

最终效果：

```

#include "highlighter.h"
Highlighter::Highlighter(QTextDocument *parent, QString extensionName)
: QSyntaxHighlighter(parent), highlightType(extensionName)
{
    if (highlightType == "cpp") /*c++代码高亮*/
    {
        HighlightingRule rule;

        /*长注释 特殊处理*/
        multilineCommentFormat.setForeground(Qt::red);
        commentStartExpression = QRegularExpression("/\\*");
        commentEndExpression = QRegularExpression("\\*/");

        /*函数*/
        functionFormat.setFontItalic(true);
        functionFormat.setForeground(Qt::blue);
        rule.pattern = QRegularExpression("\\b[A-Za-z0-9_]+(?:\\()");
        rule.format = functionFormat;
        highlightingRules.append(rule);

        /*保留字*/
        keywordFormat.setForeground(Qt::darkBlue);
        keywordFormat.setFontWeight(QFont::Bold);
        QStringList keywordPatterns;
        keywordPatterns << "\\basm\\b" << "\\bautob\\b" << "\\bbool\\b"
            << "\\bbreak\\b" << "\\bcase\\b" << "\\bcatch\\b"
            << "\\bchar\\b" << "\\bclass\\b" << "\\bconst\\b"
            << "\\bconst_cast\\b" << "\\bcontinue\\b" << "\\bdefault\\b" << "\\bdelete\\b" << "\\bdo\\b"
            << "\\bdouble\\b" << "\\bdynamic_cast\\b" << "\\belse\\b" << "\\benum\\b" << "\\bexplicit\\b"
            << "\\bexport\\b" << "\\bextern\\b" << "\\bfalse\\b" << "\\bfloat\\b" << "\\bfor\\b"
            << "\\bfriend\\b" << "\\bgoto\\b" << "\\bif\\b" << "\\binline\\b" << "\\bint\\b"
            << "\\blong\\b" << "\\bmutable\\b" << "\\bnamespace\\b" << "\\bnew\\b" << "\\boperator\\b"
            << "\\bprivate\\b" << "\\bprotected\\b" << "\\bpublic\\b" << "\\bregister\\b" << "\\breinterpret_cast\\b"
            << "\\breturn\\b" << "\\bshort\\b" << "\\bsigned\\b" << "\\bsizeof\\b" << "\\bstatic\\b" << "\\bstatic_cast\\b"
            << "\\bstruct\\b" << "\\bswitch\\b" << "\\btemplate\\b" << "\\bthis\\b"

```

5. 开发体会&小结

5.1 个人体验

组长-蒋仕彪

整个项目虽然有点赶，但我们还是完美地实现了中期报告中预计的那些功能。本来以为五个人的组效率会低，其实配合得还行，做出来效果也特别好，大家特别有成就感。

我们通过线下交流和 QQ 交流的方式来沟通，用 GitHub 完成代码管理，体验很好。我们的步骤主要是，刘书含同学先负责 UI 的设计与美化，制定一些命名规范，然后把一些事件啊按钮啊传递给我们，剩下的同学负责各种代码的设计。

因为是 OOP 的大程，我们在函数的声明与定义、函数名和变量名的命名等方面都十分谨慎，常常在纠结一个接口应该怎么取名，首字母要不要大写……哈哈感觉在代码规范方面做了很多的努力，这也是和其他课大程不一样的地方。

希望大家在体验了这次大程的合作后，能有各自不同的收获。

组员-刘书含

本次项目中主要承担了 ui 的设计和美化，比较困难的事情是 Qt 自带的一套样式表设置规范，后来采用了 designer 搭建框架，加 qss 语法描述样式完成美化的方式。由于项目的整体框架存在许多子窗口的继承，ui 在完成 QSS 选择器时需要注意选择器描述的准确性。让我感到非常有趣的是信号槽机制，在成员对象（子窗体）和父对象（父窗体）之间传递交互信息。

总体合作非常愉快，大家各自完成工作的效率也很高。特别在讨论具体实现的时候，都有自己的想法，最后采取最优秀的方法完成实现。我们还对多文件操作、代码补全进行了讨论，知道了实现功能的流程但由于太复杂搁浅了。

最后的呈现效果基本达到了预期，非常具有成就感，合作愉快~

组员-刘明锐

我本次主要负责高亮部分，我猜是因为大家不想学正则表达式？总之大部分高亮其实可以用正则表达式表示出来，而且非常简单。碰到的最严重的问题是 python 的多行注释，由于表示开始和结束都是```，所以出现```之类情况会有歧义。参考了一下 sublime，visual studio 等等的高亮，发现他们处理这个情况方法都不一样，而且很有问题……最后自己实现了强制不允许重叠，效果很好。

合作代码对接的时候发现了 Highlighter 不能实时切换高亮，进行了一些补丁。补丁写的很快，大概是因为主代码结构比较有序，而且我们接口比较简单，没出现复杂的问题。

本次项目总体感觉还是一帆风顺的，有队友协助代码的感觉也很好。

组员-李广林

这次作业我和同寝室的两位成员负责接口的定义和实现，在整个过程中从零开始学习 QT，一步一步的实现了各个模块，虽然没有太多思维上的复杂度，但在学习和实现上还是花费了很大的功夫。

这次的小组合作是非常愉快的一次合作，大家互相配合，在分完任务后都会及时的完成任务。在代码实现中，代码规范是非常要引起注意的一个地方，因为之前不太重视代码规范，所以在编写代码的时候一不注意就会定义非常随意的变量名，这也是之后要改进的地方。

总的来说，我们小组实现了预期的功能，但是有一些功能过于复杂，限于精力与时间因素没有实现，稍微有一些遗憾。

组员-吕耀维

首先非常感谢在各位大哥的带领下顺利地完成大程，从各位身上可以学习到很多东西。

由于自己的知识和能力都比较差，一开始学习 github 的用法，环境的配置以及 QT 库里各种机制时都遇到了非常大的困难，但是在大家的指导下已经能初步掌握各种软件和各种库的用法。

在这次大程中主要是实现了关于编译运行部分的操作，这部分的内容比较简单。但通过这次大程还是学习到了很多关于 c++语言的知识本身以及关于面向对象编程的一些思想。经过这次大程之后，感觉代码能力有了非常大的提高。

最后，再次感谢在大家的帮助下，自己水平得到了进步。

5.2 小组总结

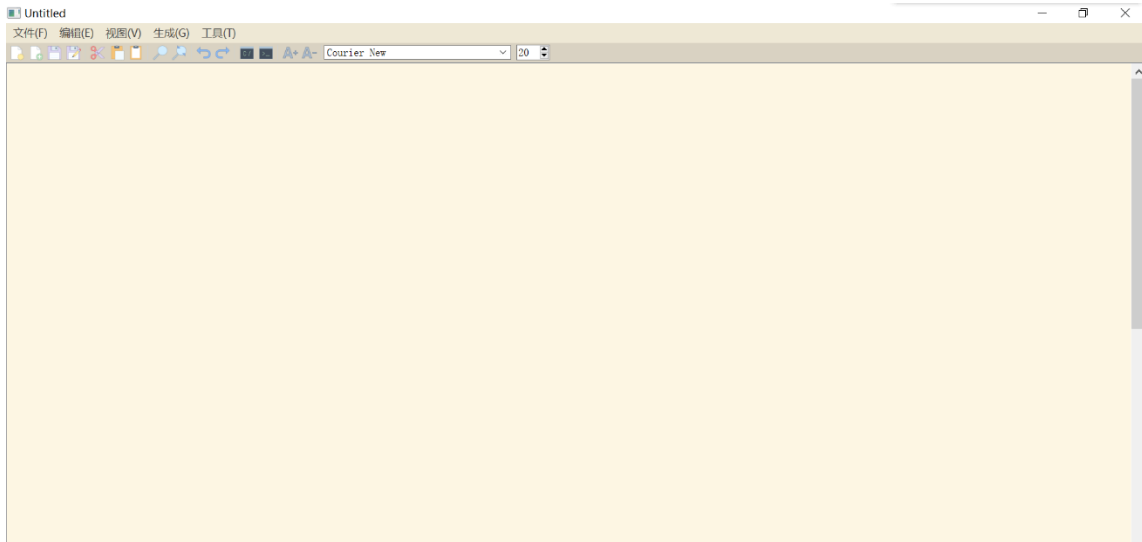
在这次的大作业项目实践中，我们将面向对象的程序设计思想广泛地运用到了项目当中，对面向对象的编程哲学有了更深的体会。

我们在实践中体会到了 C++面向对象编程中语言特性以及编程规范对于实际工程的背后的重要意义，对相关编程语言知识点有了更深刻的体会；我们全员接触、了解并使用了以 Qt 框架为基础的开发工具链，对现代 GUI 程序的开发模式以及与标准 C++略有不同的面向对象编程范式。

大家通过这样的 project 受益匪浅。

6. 测试报告

6.1 使用说明



其实使用说明已经体现在这张截图里了。

用户可以：

- 自由地在文本框里打代码。
- 修改字号和字体。
- Cpp 和 Py 文件会提供好看的高亮。
- 在菜单栏选择相应的操作。
- 在工具栏快捷选择对应的功能。
- 常用快捷键也是支持的，例如 `ctrl+C`, `ctrl+V`, `ctrl+X`, `ctrl+O` 等等
- 进行文件的保存和打开操作。

6.2.1 Cpp 的高亮（结果：正确）

6.2.1 Cpp 的高亮（结果：正确）

```
highlighter.cpp
文件(F) 编辑(E) 视图(V) 生成(G) 工具(T)
[Icons] [A·A] Courier New 10
#include "highlighter.h"

Highlighter::Highlighter(QTextDocument *parent, QString extensionName)
    : QSyntaxHighlighter(parent), highlightType(extensionName)
{
    if (highlightType == "cpp") /*c++代码高亮*/
    {
        HighlightingRule rule;

        /*长注释 特殊处理*/
        multilineCommentFormat.setForeground(Qt::red);
        commentStartExpression = QRegularExpression("/\\*");
        commentEndExpression = QRegularExpression("\\*/");

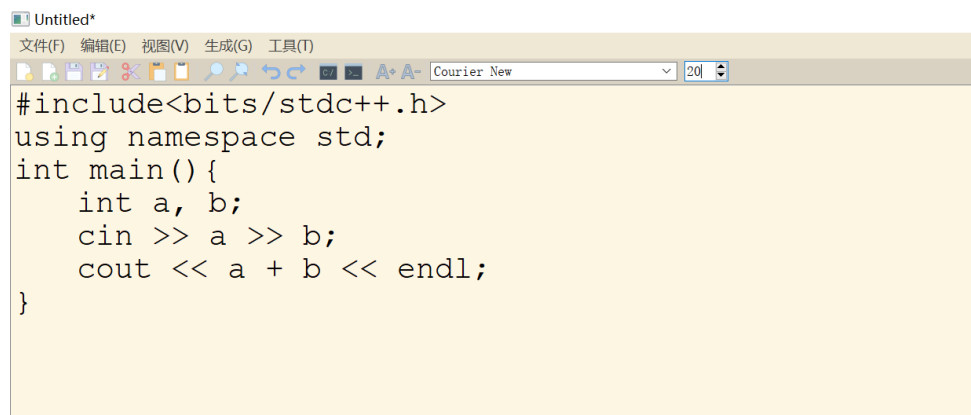
        /*函数*/
        functionFormat.setFontItalic(true);
        functionFormat.setForeground(Qt::blue);
        rule.pattern = QRegularExpression("\\b[A-Za-z0-9_]+(?=\\(|\\))");
        rule.format = functionFormat;
        highlightingRules.append(rule);

        /*保留字 写不下去了*/
        keywordFormat.setForeground(Qt::darkBlue);
        keywordFormat.setFontWeight(QFont::Bold);
        QStringList keywordPatterns;
        keywordPatterns << "\\basm\\b" << "\\bauto\\b" << "\\bbool\\b" <<
            << "\\bbreak\\b" << "\\bcase\\b" << "\\bcatch\\b" <<
            << "\\bchar\\b" << "\\bclass\\b" << "\\bconst\\b" <<
            << "\\bconst_cast\\b" << "\\bcontinue\\b" << "\\bdefault\\b" << "\\bdelete\\b" << "\\bdo\\b" <<
            << "\\bdouble\\b" << "\\bdynamic_cast\\b" << "\\belse\\b" << "\\benum\\b" << "\\bexplicit\\b" <<
            << "\\bexport\\b" << "\\bextern\\b" << "\\bfalse\\b" << "\\bfloat\\b" << "\\bfor\\b" <<
            << "\\bfriend\\b" << "\\bgoto\\b" << "\\bif\\b" << "\\binline\\b" << "\\bint\\b" <<
            << "\\blong\\b" << "\\blonglong\\b" << "\\bnamespace\\b" << "\\bnew\\b" << "\\boperator\\b" <<
            << "\\bprivate\\b" << "\\bprotected\\b" << "\\bpublic\\b" << "\\bregister\\b" << "\\breinterpret_cast\\b" <<
            << "\\breturn\\b" << "\\bshort\\b" << "\\bsigned\\b" << "\\bsizeof\\b" << "\\bstatic\\b" << "\\bstatic_cast\\b" <<
            << "\\bstruct\\b" << "\\bswitch\\b" << "\\btemplate\\b" << "\\bthis\\b" <<
            << "\\bunsigned\\b" << "\\bunsigned\\b" <<
            << "\\blonglong\\b" << "\\btypedef\\b" << "\\btypeof\\b" <<
            << "\\btemplate\\b" << "\\btypedef\\b" << "\\btypename\\b" <<
            << "\\bunion\\b" << "\\bunsigned\\b" << "\\bvirtual\\b" <<
    }
```

6.2.2 搜索的实现（结果：正确（注意 rule 已经被选中））

The image shows a Qt Creator IDE window with a C++ file named 'highlighter.cpp'. The code defines a 'Highlighter' class that inherits from 'QSyntaxHighlighter'. The code includes 'highlighter.h' and implements methods to set the parent, extension name, and highlight type. The 'highlighter.h' file is included. The code uses Qt's text formatting classes like QTextDocument, QTextCursor, and QTextCharFormat. A 'Dialog' window is open in the foreground, showing a search for 'rule' and options for case sensitivity and starting from the beginning.

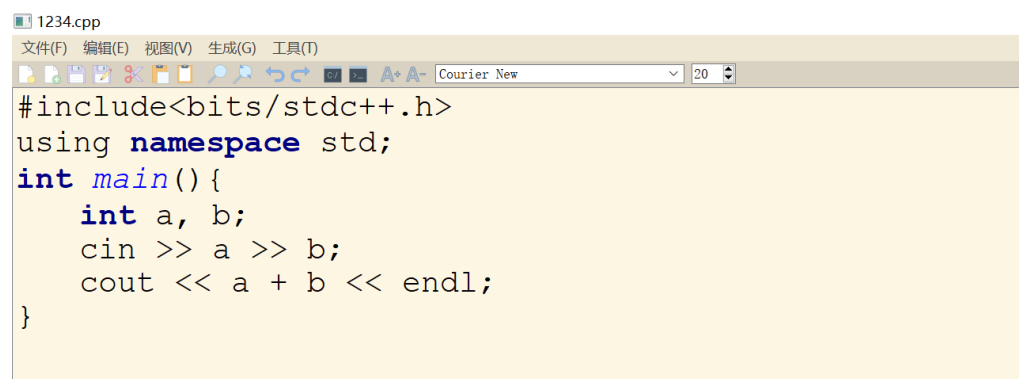
6.2.3 新建文件（结果：正确）



The screenshot shows a text editor window with a menu bar (文件(F), 编辑(E), 视图(V), 生成(G), 工具(T)) and a toolbar. The font is set to Courier New and the size to 20. The code is as follows:

```
#include<bits/stdc++.h>
using namespace std;
int main(){
    int a, b;
    cin >> a >> b;
    cout << a + b << endl;
}
```

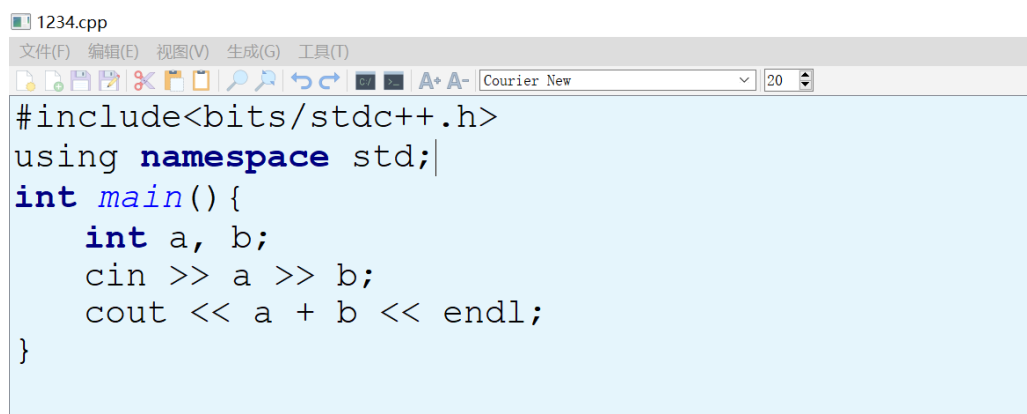
6.2.4 文件保存（结果：正确）



The screenshot shows a text editor window titled '1234.cpp' with the same menu bar and toolbar as the previous image. The code is the same as in the previous image, but with syntax highlighting: keywords are in blue and standard identifiers are in black.

```
#include<bits/stdc++.h>
using namespace std;
int main(){
    int a, b;
    cin >> a >> b;
    cout << a + b << endl;
}
```

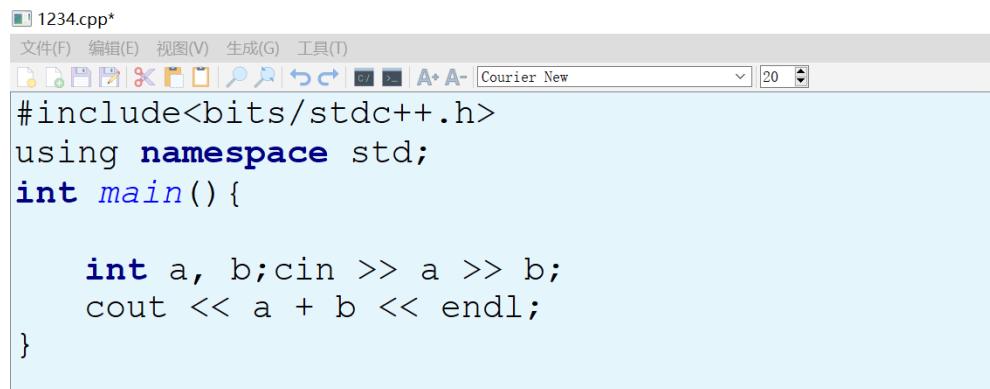
6.2.5 切换主题（结果：正确）



The screenshot shows the same text editor window titled '1234.cpp', but with a light blue background theme. The code is the same as in the previous images, with syntax highlighting.

```
#include<bits/stdc++.h>
using namespace std;
int main(){
    int a, b;
    cin >> a >> b;
    cout << a + b << endl;
}
```


6.2.6 文件的剪切，粘贴（结果：正确）

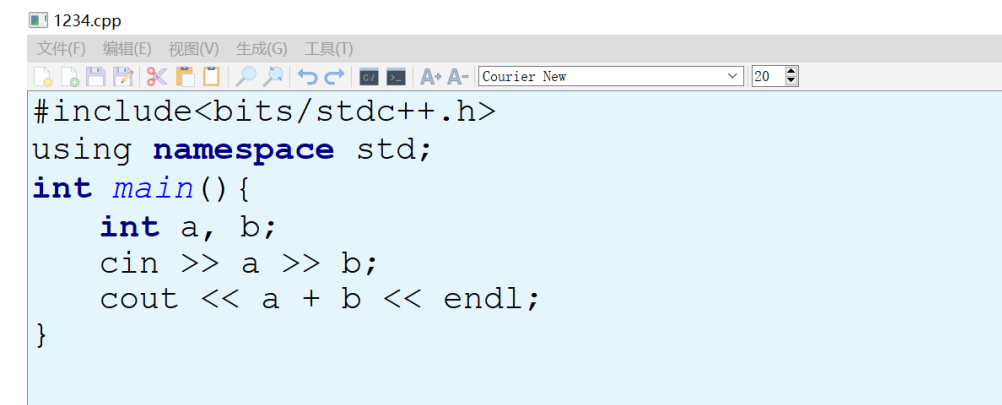


The screenshot shows a code editor window titled "1234.cpp*". The menu bar includes "文件(F)", "编辑(E)", "视图(V)", "生成(G)", and "工具(T)". The toolbar contains icons for file operations and editing. The font is set to "Courier New" and the size is "20". The code is as follows:

```
#include<bits/stdc++.h>
using namespace std;
int main() {

    int a, b;cin >> a >> b;
    cout << a + b << endl;
}
```

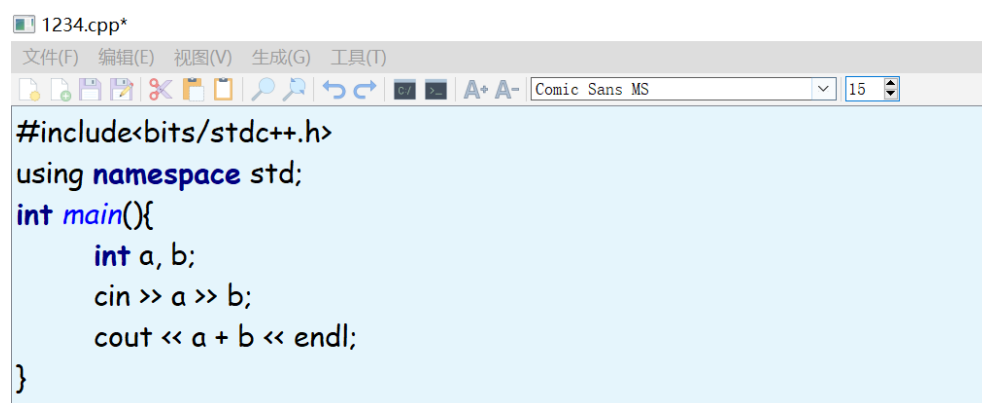
6.2.7 撤销操作（结果：正确）



The screenshot shows a code editor window titled "1234.cpp". The menu bar includes "文件(F)", "编辑(E)", "视图(V)", "生成(G)", and "工具(T)". The toolbar contains icons for file operations and editing. The font is set to "Courier New" and the size is "20". The code is as follows:

```
#include<bits/stdc++.h>
using namespace std;
int main() {
    int a, b;
    cin >> a >> b;
    cout << a + b << endl;
}
```

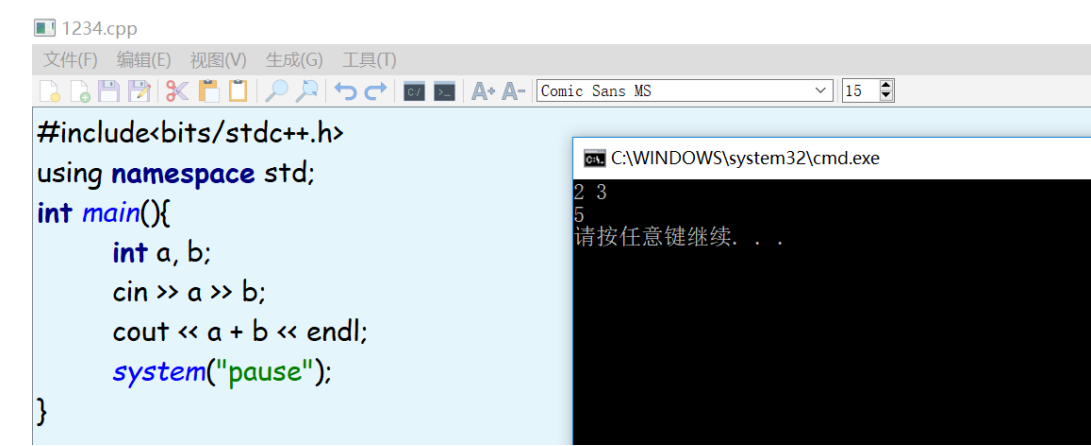
6.2.8 字号和字体的调节（结果：正确）



The screenshot shows a code editor window titled "1234.cpp*". The menu bar includes "文件(F)", "编辑(E)", "视图(V)", "生成(G)", and "工具(T)". The toolbar contains icons for file operations and editing. The font is set to "Comic Sans MS" and the size is "15". The code is as follows:

```
#include<bits/stdc++.h>
using namespace std;
int main(){
    int a, b;
    cin >> a >> b;
    cout << a + b << endl;
}
```

6.2.9 编译和运行（结果：正确）



7. 组内成员分工及互评

姓名+互评	学号	分工
蒋仕彪 3	3170102587	✧ 函数主要函数和功能实现 ✧ 总体策划，协调项目进度 ✧ 参与期中报告的书写，期末报告主要撰写者
刘明锐 3	3170105696	✧ 参与函数主要功能实现 ✧ 负责语法高亮部分详细逻辑 ✧ 参与期中和期末报告的书写
刘书含 3	3170105283	✧ UI 的约定和设计 ✧ 前端界面的美化 ✧ 督促组员跟进项目进度 ✧ 汇总和撰写期中报告，参与期末报告的书写
吕耀维 3	3170104185	✧ 参与程序编写 ✧ 程序功能测试 ✧ 参与期中报告的书写
李广林 3	3170104648	✧ 参与程序编写 ✧ 程序功能测试 ✧ 参与期中报告的书写，绘制期末报告的逻辑图

8. 附录：参与度与小组讨论平台

8.1 进展关键时间线

2019 年 2 月 26 日 小组讨论群创建，我们对组队及课程 project 进行了一定讨论，但由于没有参考资料，未有任何结论和进展。



2019 年 3 月 1 日 第二次线上讨论，因为已经有了课程要求，确定了可能的方向是画图软件或者文本编辑器。完成资料收集分工，让所有人从各方收集多种资料，最终确定完成的项目。

2019-03-01

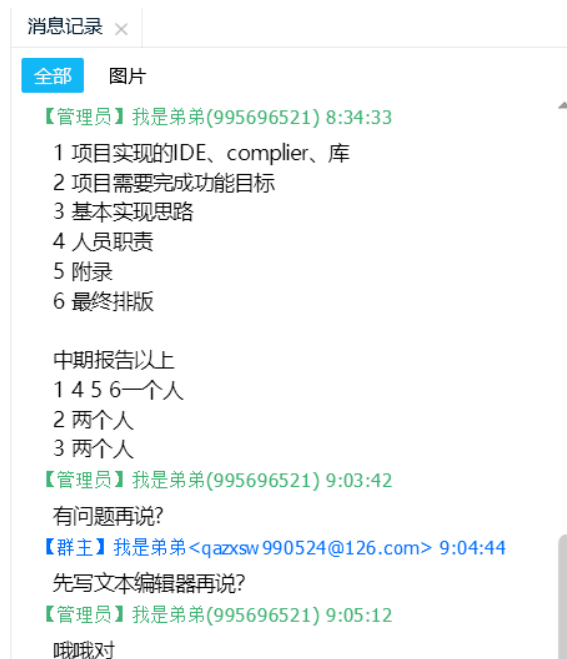
【组长】我是弟弟<heltion@qq.com> 11:34:32
不懂能做什么
【管理员】我是弟弟(995696521) 11:34:46
画图软件
【管理员】我是弟弟(995696521) 11:34:52
ide
【管理员】我是弟弟(995696521) 11:34:56
还有啥
【组长】我是弟弟<heltion@qq.com> 11:35:24
画图软件:画图 ide:记事本
【管理员】我是弟弟(995696521) 11:41:39
稳

2019 年 3 月 19 日 第一次线下讨论，交换了资料，并且确定了最终需要完成的项目为文本编辑器，扩展功能为高级文本编辑器，高级功能实现 IDE。



2019年4月5日第三次线上讨论，确定了中期报告的分工。下图是部分聊天记录截图，本次讨论确定了开发平台、实现功能以及重要的 milestone。











- 1、确定了开发平台为VS2017community，使用Qt库，编译器为MSVC++2017。
- 2、确定了普通功能的具体内容包括文本选择、插入、删除、复制、剪切、粘贴、字号调整、字体调整、颜色调整、查找、替换等。
- 3、确定了扩展功能包括代码高亮、主题配色显示。
- 4、确定了高级功能包括代码补全、编译运行、多文件操作等。
- 5、确定了中期报告的最终分工：第一部分项目实现的 IDE、Complier 及库（蒋仕彪、吕耀维），第二部分项目实现的功能（刘书含），第三部分项目实现的基本思路（蒋仕彪、刘明锐、李广林），第四部分人员分工及第五部分附录（刘书含）。



2019 年 4 月 22 日 第二次线下讨论，完成了中期报告最终的撰写，并对项目内容进行了进一步细化分工。完成了环境的统一配置，并对后半学期的重要时间节点做出了统一规划。



2019 年 5 月 30 日 线下再一次完成后续功能实现，合并代码

 SubCode 1.2.0.zip	2019/5/30 22:56	36
 SubCode0.0.1.zip	2019/5/30 14:58	36
 SubCode1.0.1.zip	2019/5/30 20:20	36
 SubCode1.1.0.zip	2019/5/30 22:06	36
 SubCode1.1.1fontsizewrong.zip	2019/5/30 22:06	36
 SubCode1.3.0.zip	2019/5/30 23:17	36
 SubCode1.5.0.zip	2019/6/3 10:45	36
 SubCode1.6.2.zip	2019/6/22 14:38	36
 SubCode1.6.3(ui changed).zip	2019/6/25 20:57	36
 ..		



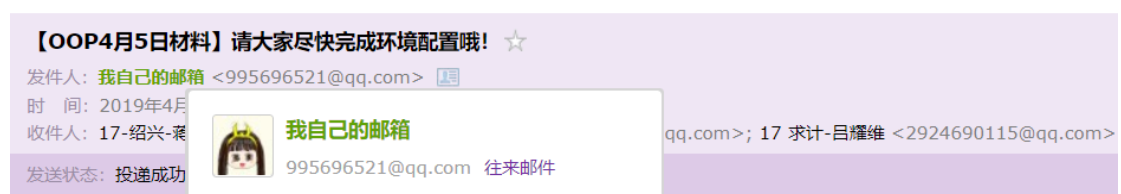
此次完成到 1.3 版本，实现了基础的全部功能及部分高级功能，但仍然存在 bug。

2019 年 6 月 3 日，线上合作，完成到 1.5.0 版本，修复了启用高亮就文件保存的 bug，下一版本预计实现 search 和 replace 功能。

2019 年 6 月 22 日，线下最后验证功能并合成 PPT。

8.2 在线合作记录

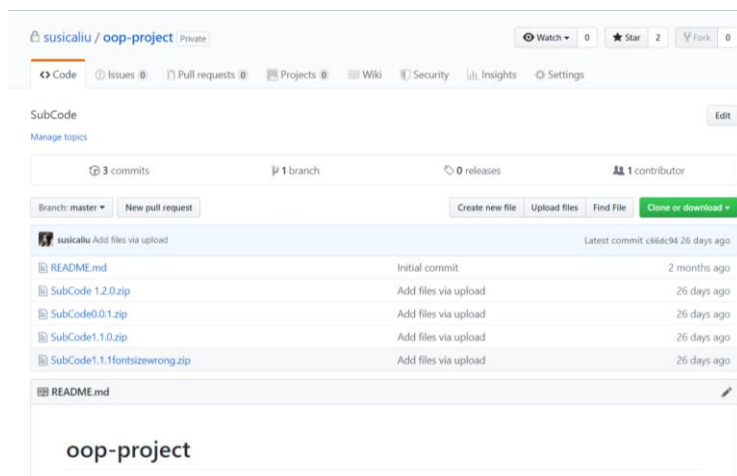
8.2.1 邮件记录



- 1、下载VS2017Community中的Visual C++
 - 2、下载Qt工具forVS2017
 - 3、在VS2017中安装QT Tool
 - 4、完成配置，新建Qt项目。
- :)如果还有不清楚的，指路CSDN~

8.2.2 Github 提交记录

事实上是因为有组员不会使用 git 操作，所以最后抛弃了 github 合作转而使用 QQ 传递。由于我们几乎每一次都是线下合并各自功能，所以冲突不严重，在线合作工具也不太必要。



8.2.3 QQ 群文件提交记录

