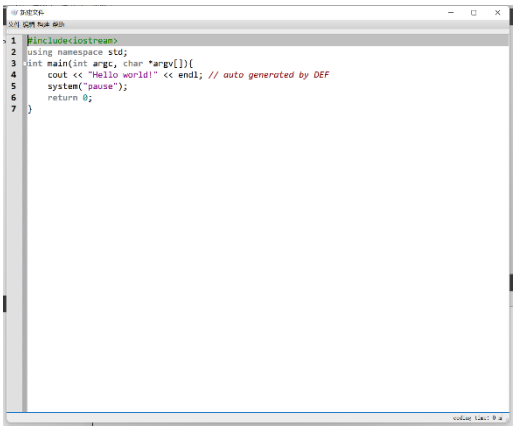
Qt 大作业——Def C++ IDE 报告

孟梓墨、胡建波、高文硕

Github 网址：https://github.com/Toseic/def-CPP-IDE

一、程序功能介绍

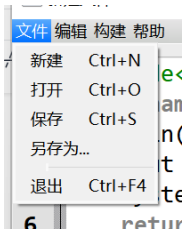
我们小组受到 Dev C++的启迪，决定使用 Qt 开发一款属于自己的 C++ IDE，并取名为 Def C++。通过几周的开发，该软件已经具有较为完善的文件系统管理功能、代码编辑功能、编译与运行功能、设置功能和其他功能。



（主界面效果图）

1、文件系统管理功能包括：

- 新建：建立一个新的文件，开启一个新的窗口，生成默认代码（如上图）。
- 打开：打开已有的一个 cpp 文件。
- 保存：保存 cpp 文件，如果是新建文件则选择保存地址。
- 另存为：选择保存地址，另存 cpp 文件。
- 退出：关闭窗口。



（文件系统管理效果图）

2、代码编辑功能包括：

- 代码编辑器：可以编辑代码，拥有基本的文本编辑功能，包括撤销、重做、复制、剪切、粘贴、全选。
- 文法分析器：分析代码中的不同部分，标注不同的颜色或粗细等。
- 侧边和高亮：代码左边的行号，以及通过大括号折叠一部分代码，高亮选中处的位置，

寻找对应括号并高亮。

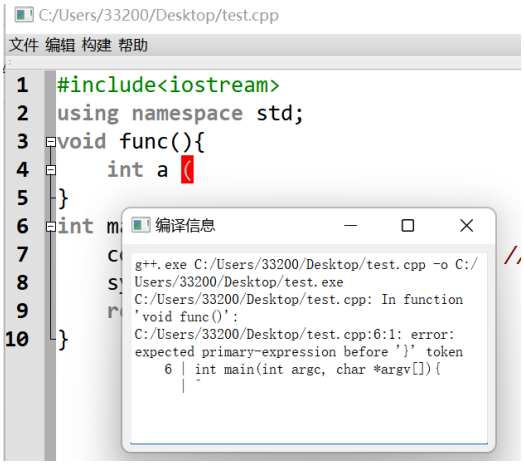
- 自动化：自动缩进，自动括号补全，自动引号补全，关键词提示与补全，标识符提示与补全（变量名、函数名等）。

```
3  void func(){
6  int main(int argc, char *argv[]){
7      cout << "Hello world!" << endl; // auto generated by DEF
8      system("pause");
9      return 0;
10 }
```

（代码折叠与高亮效果图）

3、编译与运行功能包括：

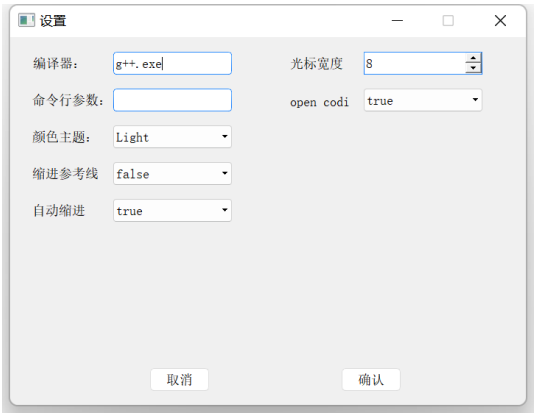
- 编译：先自动保存，然后使用 g++将 cpp 文件编译生成 exe 文件。如果出现报错，则弹出报错内容的窗口。
- 运行：执行编译的 exe 文件。
- 编译并运行：只有编译没有报错才会自动运行生成的 exe 文件。



（编译报错信息效果图）

4、设置功能包括：

- 设置编译器，默认为 g++.exe。
- 设置命令行参数，默认为空。
- 设置软件的界面颜色主题。
- 设置是否显示缩进参考线/是否自动缩进/显示 Coding time。
- 设置光标宽度。
- 保存设置到配置文件，打开时自动读取。



（设置界面效果图）

5、其他功能包括：

- 可爱的自制 logo。
- Coding time。
- 外部导入的 qss。

二、项目各模块与类设计细节

1、*main* 函数

创建一个 *MainWindow* 对象，设置窗口大小、icon、qss，然后打开窗口。

2、*MainWindow* 类

属性：

- *Texteditor *textEdit*：代码编辑器（自定义类，具体介绍见后）。
- *QString file_path, compiler, commandline_arg*：保存文件路径、编译器名字和命令行参数。
- *QString mytheme, themepath*：当前使用主题，主题保存路径。
- *settingStore* set_store*：用于设置的保存。
- *QLabel *codingtime*：Coding time 的显示。
- *QMenu* 类的多个对象：用于主界面的菜单。
- *QAction* 类的多个对象：用于菜单内的各个选项。

函数：

- *MainWindow(QWidget *parent)*：构造函数。设置标题为“新建文件”，设置默认编译器名字、默认命令行参数，更新 Coding time 的线程，在 statubar 中添加 codingtime label，在中央摆放代码编辑器。搭建各个菜单栏，包括“文件”、“编辑”、“构建”、“帮助”。在各个菜单栏里搭建选项，包括“文件”内的“新建（设置快捷键为 Ctrl+N）”、“新建（设置快捷键为 Ctrl+N）”、“打开（设置快捷键为 Ctrl+O）”、“保存（设置快捷键为 Ctrl+S）”、“另存为（设置快捷键为 Shift+Ctrl+S）”、“退出（设置快捷键为 Ctrl+W）”，“编辑”内的“撤销（设置快捷键为 Ctrl+Z）”、“重做（设置快捷键为 Ctrl+Shift+Z）”、“复制（设置快捷键为 Ctrl+C）”、“粘贴（设置快捷键为 Ctrl+V）”、“剪切（设置快捷键为 Ctrl+X）”、“全选（设置快捷键为 Ctrl+A）”，“构建”内的“编译”、“运行”、“编译并运行（设置快捷键为 F5）”、“设置（设置快捷键为 F1）”，“帮助”中的“关于（设置快捷键为 Ctrl+H）”。链接各个消息槽。
- *void set_compiler(QString)*：设置编译器。
- *void set_commandline_arg(QString)*：设置命令行参数。
- *void set_theme(QString)*：设置颜色主题。
- *void set_cursor_width(int)*：设置光标宽度。
- *void set_auto_tab(QString)*：设置换行时自动缩进。
- *void set_auto_tab(bool)*：设置换行时自动缩进。
- *void set_referline(QString)*：设置缩进参考线。
- *void set_referline(bool)*：设置缩进参考线。
- *void set_codingtime(bool)*：设置是否打开 coding time 显示。
- *void set_store_refresh()*：通过 setstore 的内容更新所有设置。
- *void on_new()*：新建文件，显示新窗口，设置新窗口的样式和图标。
- *void on_open()*：打开文件，先读取文件位置，如果读取成功，则将前面提到的属性 file_path 进行修改，同时对窗口标题进行修改。读取对应文件内容，通过 buffer 放入代

码编辑器中。

- *void on_save()*：保存文件，先判断是新建文件还是已经打开的文件，如果是新的文件就跳转到另存为，如果是已经打开的文件，就将代码编辑器的文本内容存入当前的 file_path。
- *void on_saveas()*：另存为文件，先选择保存的位置，如果路径正确，就将代码编辑器的文本内容存入当前的 file_path。
- *void on_exit()*：直接粗鲁地关闭窗口！
- *void on_undo()*：撤销！
- *void on_redo()*：重做！
- *void on_selectall()*：全选！
- *void on_copy()*：复制！
- *void on_cut()*：剪切！
- *void on_paste()*：粘贴！
- *bool on_compile()*：编译，先调用保存，然后产生一个“生成路径”，是把 cpp 后缀换成了 exe，通过属性编译器名字生成一条编译指令（compiler + " " + file_path +" -o " + generate_file_path）。在 QProcess 中运行该指令，如果有报错信息，建立一个新的编译信息窗口进行显示，返回 false，否则返回 true。
- *void on_run()*：运行，生成一条含有命令行参数的运行指令（generate_file_path+" "+commandline_arg），并且运行。
- *void on_compilt_and_run()*：如果编译成功就运行！
- *void on_settings()*：打开设置界面，并且为这个界面也设置好颜色主题。
- *void on_about()*：打开“关于”界面。

3、*Texteditor* 类

注：关于使用外部库 *QScintilla* 的说明：

为了更好的达到我们对软件设计的目标，我们导入了一个叫做 *QScintilla* 的外部库，通过我们的研究，这个库可以实现文法分析、自动补全等功能。使用这个库并不代表我们在代码编辑方面没有工作量，因为这个库需要我们自己设置各项功能，自己设计各部分代码的颜色，以及自动括号/引号补全是我们自己写函数实现的。在代码中，我们放置了名为 QScintilla 的文件夹，文件夹内包含 qscintilla2_qt5.dll、qscintilla2_qt5d.dll 和 Qsci 文件夹三项内容，由于环境不同，在个别编译环境下 dll 会无法使用，此时可以自行下载 QScintilla，编译出两个 dll 文件，进行替换即可。

Texteditor 类继承了 *QsciScintilla* 类。

函数：

- *void setapis()*：将所有 C++关键词设置好。
- *void init_editor()*：初始化，设置好各部分的颜色和字体（经过多个小时后的配色专业研究最终选择了比较朴实无华的低调版），调整总体大小，设置光标宽度、高亮显示光标所在行、括号匹配、代码折叠、UTF-8 编码、补全和提示。填入默认代码。连接消息槽，每次有文本变化就看看是否该补全括号、引号。

（默认代码即：″#include<iostream>\nusing namespace std;\nint main(int argc, char *argv[]){\n\tcout << \"Hello world!\" << endl; // auto generated by DEF\n\tsystem(\"pause\");\n\treturn 0;\n}″）

- *void complete_brackets()*：补全括号，如果检测到光标处的是“（”，就放置一个“）”，然后把光标移到中间。这一点应用到“[]”完全没有问题，但是应用到“{}”就出现新的问题，我们想做到输入“{”时能自动换行，所以我们记录了 Tab 的数量，然后加入了自动换行，自动纳入 Tab+1 个 Tab，这样就达到了快捷换行的效果。应用到″″和′′的

时候又出现了新的问题，因为前后引号都是一样的，所以他会一直不断生成，为了解决这个问题，我们加入了 bool 判定，每输入一个引号只能出来一个引号。

4、Settings 类

首先布置了一个界面，见总体介绍里面的图片。

函数：

- *Settings(MainWindow *set, QWidget *parent)*：构造函数。首先找到自己的亲生父亲，这样设置的时候才知道设置的是谁。然后把自己的 UI 连接到各个槽函数。
- *void on_cancel_clicked()*：取消，啥也不做直接关闭窗口！
- *void on_confirm_clicked()*：确认，完成各项设置，这里调用的是 MainWindow 的 set 系列函数。

5、*setting store* 类：

用于储存设置信息，储存的设置包括：颜色主题设置、光标宽度、是否自动缩进、缩进参考线等，在类的构造函数中会把设置先设置为默认的参数，然后尝试读取配置文件，读取后将配置文件的内容覆盖默认值。

而在程序窗口关闭时，这个类会自动将储存的配置储存到一个配置文件中，以供下次打开程序时读取。

setting store 类在实例化后会被储存在 *mainwindow* 中，在 *mainwindow* 开启了 setting 窗口供用户进行设置后窗口关闭时，会先由 setting 调用 *mainwindow* 修改 *settingstore* 的内容并更新储存文件的内容，然后再由 *mainwindow* 调用相应函数使得 *settingstore* 的配置实际生效。

6、coding time 相关的类： *ct_thread*

作为一个守护线程运行，每隔一分钟更新 *mainwindow* 里面的 *coding_time* 这个 label 的内容。

三、小组成员分工情况

高文硕：*MainWindow* 类的编写，*QScintilla* 库的使用，*TextEditor* 类的优化，编译信息框，*Settings* 类的初步编写，搞了个 icon 和 qss。

孟梓墨：*TextEditor* 类的编写以及各项的设置、实现括号补全功能，debug 功能的编写（已废弃）。

胡建波：*Settings* 类、*setstore* 类的编写，各项设置功能、Coding time 功能、将设置存储到配置文件功能的编写。

四、项目总结与反思

相比于为更多人所选择的游戏等项目，我们组的 def-C++ IDE 项目更具有挑战性。起初在确定项目方向的时候，高文硕同学提了一句“有没有一个做 C++ IDE 的可能性”，大家觉得这个想法有一定的可行性于是就开搞了。我们先基于 Qt 做了一个基本框架，通过调用 g++.exe 完成了编译运行功能。之后摆在我们面前的问题是如何对文本编辑部分进行美化。如果使用 Qt 自带的 *textedit* 类进行个性化设计的话，不管是工作量还是学习成本都超出了我们组的能力范围。正当项目进度难以推进时，我们找到了一个外部库 *QsciScintilla*，我们之后的大部分工作都基于 *QsciScintilla* 进行。我们设计了一个 *QsciScintilla* 的派生类 *TextEditor*，将其作为项目的文本编辑部分。在此之外，我们还设计了编辑器的颜色主题，添加了 coding time 等功能。由于时间仓促，查找替换功能并未开发，断点调试功能还在开发中，所以最后提交的 def-C++ IDE 项目并没有这两个功能。

我们项目的代码量总的来说并没有很高，仅有 1000 行左右。但是由于我们使用了外部

库 *QsciScintilla*，再加上 C++ IDE 项目的特殊性，对从来没有接触过 Qt 编程以及 C++ IDE 底层实现原理的我们组来说，研究现有项目代码以及资源的学习成本显然是更高的——实际上，这些工作也占用了我们组的大部分时间，甚至是 *QsciScintilla* 的安装也给我们造成了困难，网上有关 *QsciScintilla* 的资料本就很少，由于各种博客之间的抄袭关系，事实上的有效信息就更少了。通过不断地摸索，我们克服了许多困难，最终完成了 2022 程序设计实习的 Qt 项目大作业。虽然该项目仍然显得有些粗糙，但是我们仍然对它感到很满意。

在项目的开发过程中，我们也得到了很多宝贵的经验与收获。首先是对于 C++程序编译运行以及我们经常使用的断点调试功能的深入理解，在大一阶段的其他两门专业课《计算概论 A》、《人工智能引论》中，对于代码我们仅仅是机械地在执行“编译-debug-运行-debug”的循环，而对于我们使用的 C++ IDE 的工作原理并没有了解和认识。在这次开发 IDE 的编译运行和断点调试功能的过程中，我们了解到了 g++.exe 和 gdb.exe 的作用以及一些使用方法，而在其他各种功能的开发过程中，通过 Qt 的信号与槽机制，我们也理解了从代码到可执行的程序之间各种“按钮”与行为之间的响应关系。其次是对程序设计实习上半学期学习的 C++面向对象程序设计的复习，Qt 编程给了我们很好的练习机会。最后是小组合作和 github 使用的宝贵经验，在这次项目中，我们组使用 github 进行版本管理，熟悉了 git 相关操作以及应用，为之后的项目开发打好了基础。

感谢刘家瑛老师、胡煜章助教给我们的指导，同时也感谢程序设计实习教学组安排的这次大作业，我们在这次大作业中受益良多。

参考文献：

[1] *QsciScintilla* 的官方文档：https://docs.huihoo.com/pyqt/QScintilla2/index.html