浙江大学

程 序 设 计 专 题

大 程 序 报 告

大程名称： Lyther 轻羽 - 轻量级 C 语言编辑器

指导老师： 刘新国

2022~2023 春夏学期 2023 年 5 月 23 日

报告撰写注意事项

1） 图文并茂。文字通顺，语言流畅，无错别字。

2） 书写格式规范，排版良好，内容完整。

3） 存在拼凑、剽窃等现象一律认定为抄袭；0 分

4） 蓝色文字为说明，在最后提交的终稿版本，请删除这些文字。

目 录

轻羽 - 轻量级 C 语言编辑器大程序设计项目

1 大程序简介

1.1 选题背景及意义

经过简单调研，我们发现对于 C 语言初学者而言，常见的工具有两类：第一类是侧重于工程开发的 IDE，如 Dev-C++、Visual Studio 等。它们往往具有体积庞大、操作复杂、学习曲线较为陡峭等诸多缺点。另一类是侧重于文本编辑的代码编辑器，如 Visual Studio Code, Atom 等。这类编辑器较为轻便，但也需要进行一定的配置，同时由于需要安装，缺失了一定的便携性。

在这一背景下，我们小组期望打造一款面向 C 语言初学者的轻量级 C 代码编辑器，做到像 Windows 自带的记事本软件一样：界面简洁，操作简单，便携小巧，即点即用，同时支持代码编辑器常有的高级功能，如代码高亮，自动补全等。

经过讨论，我们将这个项目命名为“轻羽”，“轻”寄托着我们追求的目标，“羽”象征羽毛笔，强调编辑代码的实用性。而英文名“Lyther”则取自“Light”和“Feather”之音，同样表达了我们的目标。

开发这样一款软件，既填补了面向初学者的代码编辑器的空缺，也有助于小组成员更加深入地了解 C 语言及其编译的过程。颇具挑战的任务，也能激发小组成员学习和研究的热情。

1.2 目标要求

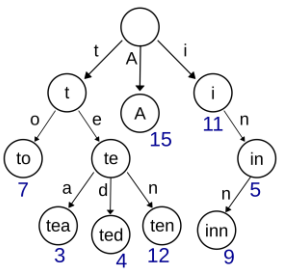
我们希望实现以下功能：

- 基本功能：
  - 支持菜单功能：文件、编辑、帮助菜单栏及对应操作。
  - 支持代码编辑器常规的编辑功能。
  - 支持当前位置光标闪烁。
- 高级功能：
  - 鼠标定位光标、鼠标左键选中字符块、鼠标右键选项（拷贝/剪切/粘贴/恢复/重做）等功能。
  - 行号显示功能。
  - 语法高亮功能。
  - 括号匹配，括号自动补全，代码自动补全。

我们希望解决以下场景中的问题：

1.3 术语说明

- **词法分析(Lexical Analysis):** 将程序源文件作为输入，并将其分解成一个个独立的词法符号，即“单词符号”(Token)。
- **单词符号(Token):** 程序设计语言中具有独立含义的最小语法单位，可以类比成自然语言中的“单词”。Token 具有不同的类型，比如变量类型，运算符，字符串，注释，括号等等。
- **字典树(Trie / Prefix Tree):** 字典树是一种有序树 [1]（如右



图），用于保存关联数组，其中的键通常是字符串。在字典树上，每一个节点代表一个字符，这个节点既可以是终止节点（代表一串字符的结束），也可以是中间节点。同时，如果两个节点有相同的父节点，就代表它们在这个字符之前有相同的前缀。

- **树的深度优先遍历：**从树的根节点出发，递归地访问每一个节点。首先访问当前节点的所有子节点，然后再返回，递归出口是当前节点没有任何子节点。

2 需求分析

2.1 功能需求

- 基本功能：
  - 支持菜单功能：
    - 文件：新建、打开、保存、另存为、关闭、退出（均含相应的快捷键）
    - 编辑：拷贝、剪切、粘贴、恢复、重做
    - 帮助：使用说明、关于
  - 支持文本输入、上下左右移动光标、Backspace 后退擦除、Delete 删除、选择/剪切/复制/粘贴/恢复/重做等键盘操作功能
  - 支持当前位置光标闪烁
- 较高级功能：
  - 鼠标定位光标、鼠标左键选中字符块、鼠标右键选项（拷贝/剪切/粘贴/恢复/重做）等功能
  - 支持行号功能
  - 语法高亮功能：包括类型、运算符、字符串、注释、括号配对、当前行特殊背景色
  - 括号匹配，括号自动补全，代码自动补全

2.2 数据需求

输入数据：

前端的图形化界面主要接受用户的两种输入，分别是键盘输入和鼠标输入:

- 键盘输入
  - 单按键输入——对应文本输入功能
    - 输入单个字符，类型为`<char>`，范围在（`'a'~'z','A'-'Z', <0`）添加在指定文本区域。
    - 输入方向键，类型为`<ACL_KEYBOARD_EVENT>`,取值范围为`<VK_LEFT,VK_RIGHT,VK_UP,VK_DOWN>`
    - 特殊输入`<VK_BACK, '\t'>`
  - 组合键输入——对应快捷键功能

`<CTRL-C, CTRL-V, CTRL-X, CTRL-Z, CTRL-U, CTRL-Y, CTRL-P,CTRL-H>`

- 鼠标输入——对应图形化操作功能
  - 点击 `<BUTTON_DOWN, BUTTON_UP>`
  - 移动 `<MOUSE_MOVE>`
  - 滚轮 `<ROLL_DOWN, ROLL_UP>`

而对于后端的代码处理模块，主要接受文件输入和剪切板输入：

- 文件输入——对应代码读写功能
  - 输入文件：约束是使用 GB2312 编码的合法的 C 语言文件（后缀名为.c 或者.h）
  - 剪切板输入：约束是使用 GB2312 编码的合法字符串。

输出数据：

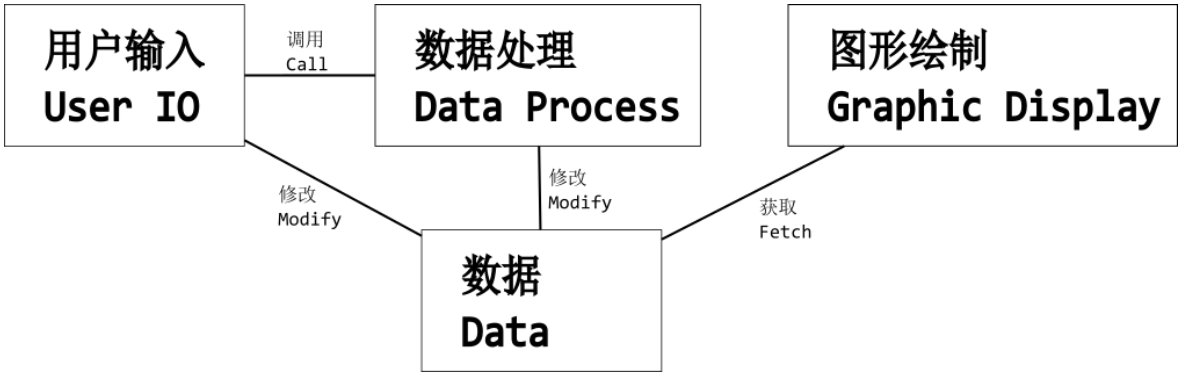
- 由代码处理模块进行文件输出

2.3 性能需求

我们预设的常规图形刷新时间为 20ms（绘制频率 50Hz），同时在有操作时进行额外刷新。考虑到代码编辑器对于刷新率需求不高，并且一般情况下人为操作间隔都大于 20ms。所以我们认为一次刷新流程的最大时间应小于 20ms，以达到流畅的操作体验。一次刷新主要分为两个步骤，第一步是获取输入，重新解析代码高亮，第二步是绘制图形。·获取输入花费的是常数级时间，时间瓶颈主要在解析代码高亮上。我们的算法采取重新解析当前行（添加或删除位置），包括将当前行拆分成多行的情形（输入含有<'\n'>）。整个解析为线性时间复杂度，即进行一次线性扫描，分割词块。其中涉及的链表操作也为线性时间。经测试，大文件打开（10000 行，每行 1000 字符）解析所花费时间为 6300ms（主要用于语法高亮）。则用户一次操作的解析可以控制在 2ms 左右。·经过我们的测试，在无代码情况下，即绘制基本窗口时，所需要的时间为 2ms。上文所述大文件打开后每次刷新所需时间 150ms（因为打开后每次只解析绘制窗口范围内的内容（21 行），所以时间会短很多）。大致计算后发现，每行控制在 100 字符左右即可达到预期的流畅度，而正常使用情况下，代码平均一行字符数小于 100，可以满足我们的性能要求。

3 程序开发设计

3.1 总体架构设计



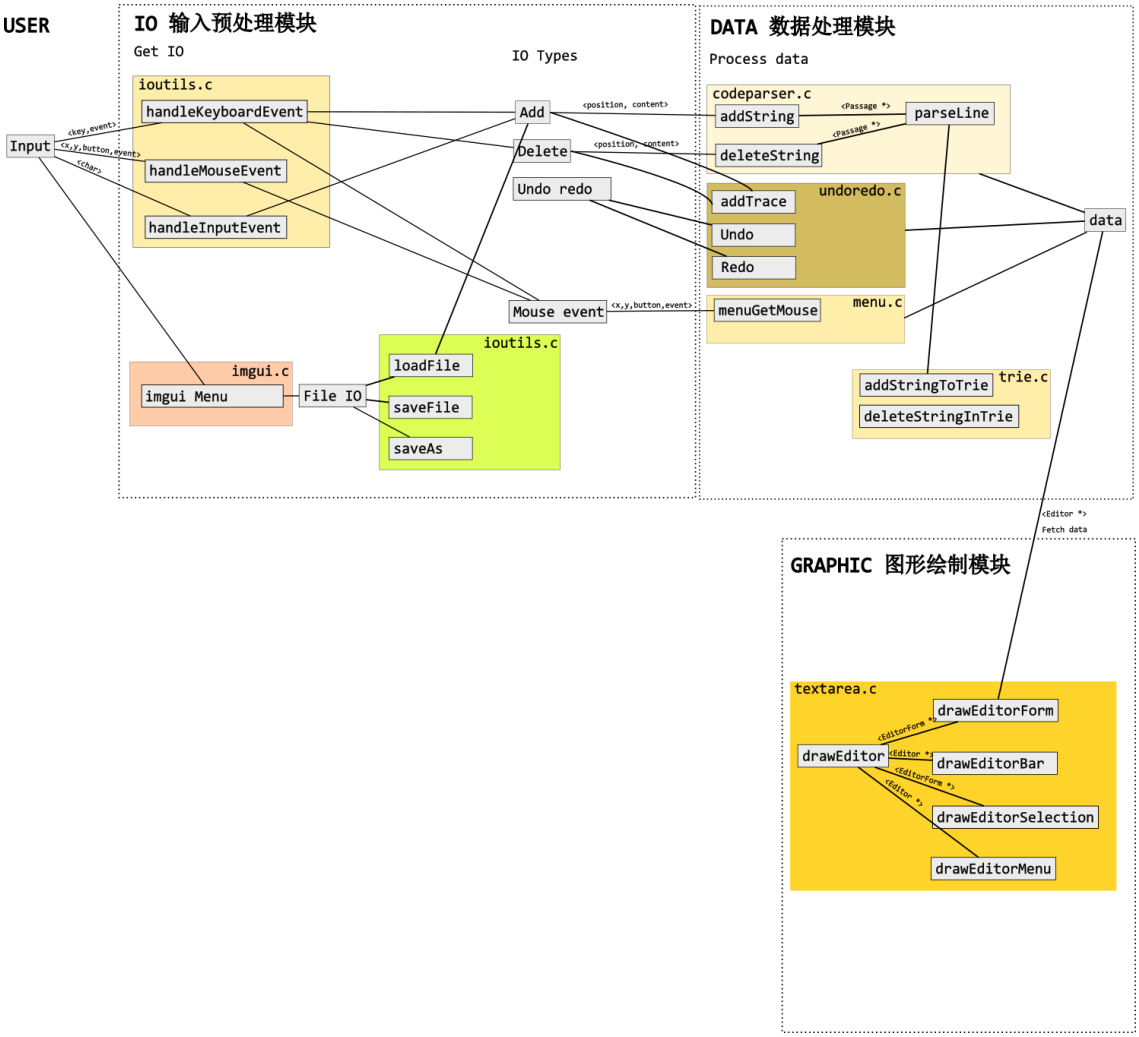
我们的总体架构分为三个部分，输入预处理模块、数据处理模块、图形绘制模块。

总体流程为：用户输入→输入模块预处理输入内容，交给数据处理模块→数据处理模块将获得的数据与原有数据进行重新整合，生成图形绘制所需的必要参

数→图形绘制模块获取绘制参数，进行绘制。

其中，图形处理模块仅负责根据相关参数进行机械化绘制，与数据处理模块分离。

各模块详细关系：



3.2 功能模块设计

3.2.1 输入预处理模块

1. 鼠标事件

```
void handleMouseEvent(Editor *editor, int x, int y,
                      int button, int event);
```

- 鼠标事件主要处理两个功能：光标定位选区与字号调整
  - 该模块用静态局部变量 `isLeftButtonDown` 和静态全局变量 `isControlDown` 记录鼠标左键和 `CTRL` 键按下的状态。
  - 鼠标左键按下后，`pixelToPosRC` 函数将鼠标坐标转换为文本坐标，将其作为光标位置。
  - 鼠标移动超过一个字符后，进入选区状态，直至下次左键按下或选区区间长度为 0，`inSelectionMode` 设置为真，并记录选取的起始和终点位置。

- 鼠标滚动后，若已按下 `CTRL` ，则相应调整静态全局变量 `textPointSize`

2. 键盘事件

```
void handleKeyboardEvent(Editor *editor, int key, int event);
```

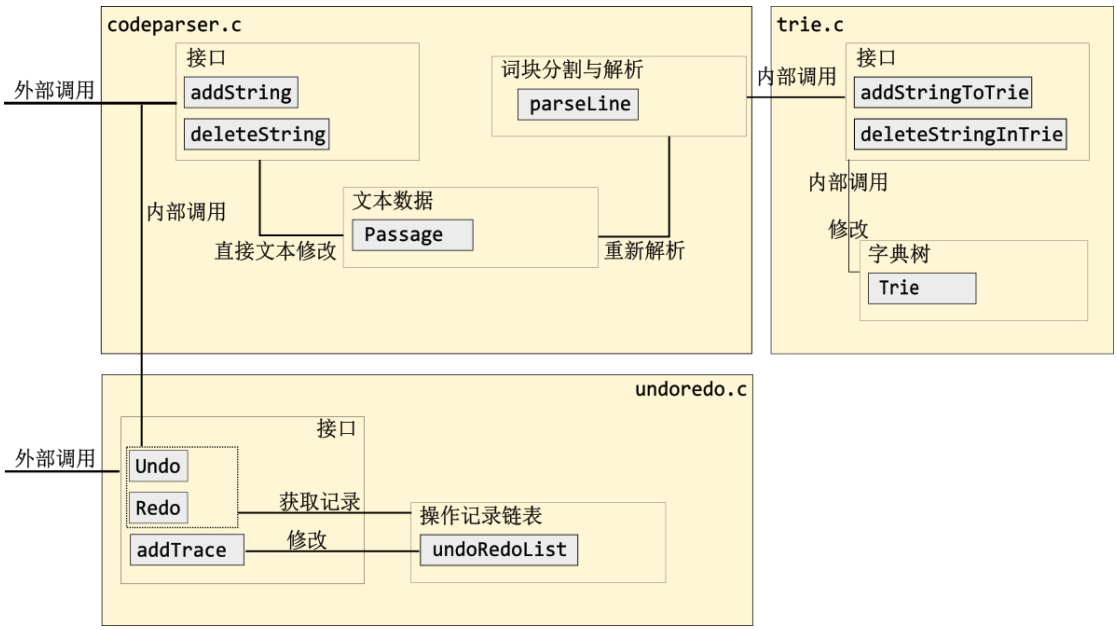
- 键盘事件处理：光标移动，退格、回车、删除键
  - 方向键被按下后，调用 `moveCaret` 函数进行相应移动
  - 退格、回车、删除按下后，调用数据处理模块的 `deleteString`，`addString` 在相应位置添加和删除，并调用 `addTrace` 记录此次操作，用于撤消重做。

3. 字符输入事件

```
void handleInputEvent(Editor *editor, char ch);
```

- 字符输入事件处理：字符的直接输入、括号自动补全。
  - 静态局部变量 `lastCn` 和 `completed`
  - 英文字符输入直接调用 `addString` 添加 `addTrace` 记录
  - 中文字符输入则先判断 `lastCn` 变量（上一个字符为英文时等于 0），如果为 0，则 `lastCn` 等于当前输入，否则将 `lastCn` 与当前输入一起（两个合在一起是一个中文字符）用 `addString` 添加 `addTrace` 记录
  - 若检测到输入字符为左括号，则自动补全，`completed` 设置为真。若下一个输入为已自动补全的输入（`completed` 为真时），则忽略此输入。

3.2.2 数据处理模块

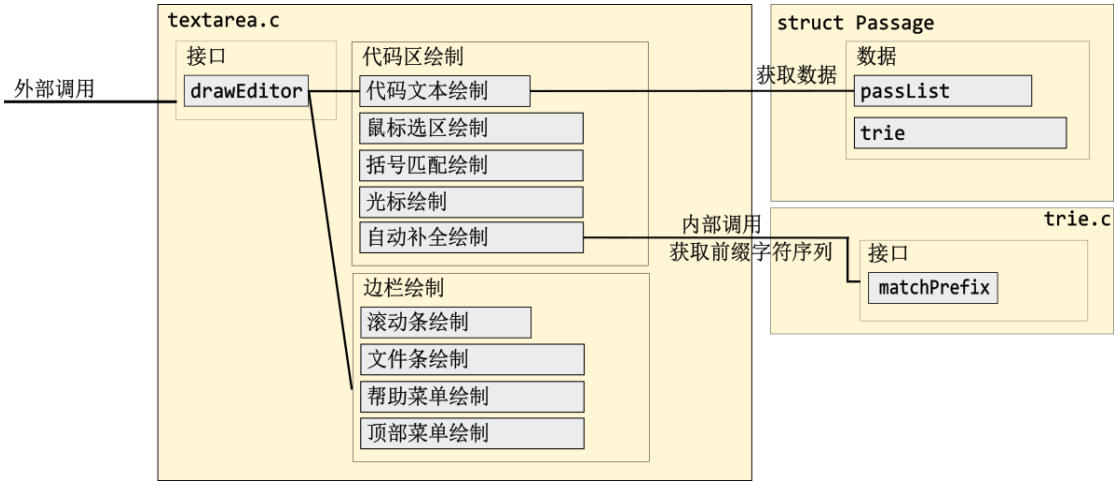


数据处理模块根据有两个外部接口，一是关于代码文本修改的接口，二是关于撤消重做的接口。

代码文本修改接口会先直接以普通文本方式操作 `Passage` 数据结构，进行相应添加与删除，随后提交给 `parseLine` 函数进行重新解析。解析完成后调用字典树接口，更新字典树。

撤消重做接口则会修改撤消重做列表，并根据相应记录，内部调用代码文本修改接口。

3.2.3 图形绘制模块



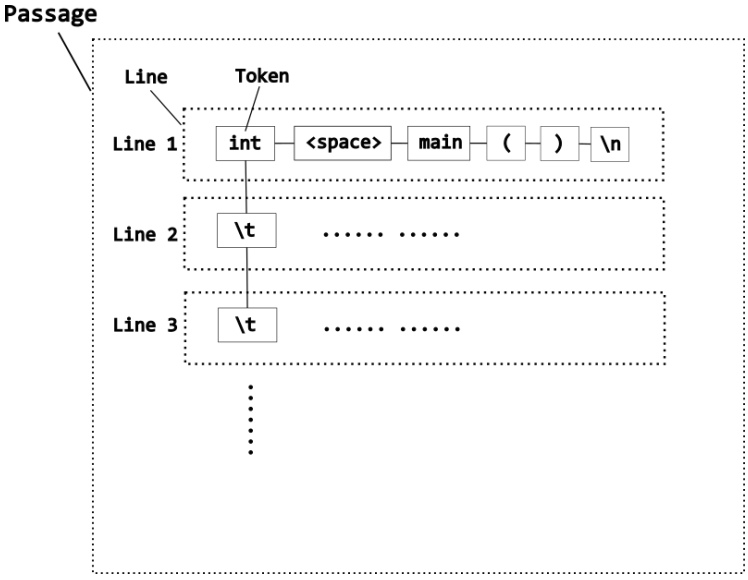
图形绘制模块主要分为两大部分：

代码区域的绘制和其它区域的绘制。代码内容的绘制会获取 `passList` 结构体中已经标记好的词块，然后根据类型设置字体样式。自动补全则调用接口，获取前缀匹配结果，然后绘制成功匹配的字符串。

边栏绘制主要有滚动条，文件选择条，菜单。各控件参数均为实时计算。

3.3 数据结构设计

1.代码内容数据结构：



我们将一行代码拆分成多个 Token，定义如下：

```
typedef struct {
    char content[MAX_WORD_SIZE];
    int length, level;
    CodeTokenType type;
} Token;
```

这个结构体记录了 Token 的内容，长度，层级和类型。其中 CodeTokenType 定义为：

```
typedef enum {
    STRING,
    COMMENT,
    LEFT_COMMENT,
    RIGHT_COMMENT,
    PREPROCESS,
    KEYWORD,
    LEFT_PARENTHESES,
    RIGHT_PARENTHESES,
    LEFT_BRACKETS,
    RIGHT_BRACKETS,
    LEFT_BRACE,
    RIGHT_BRACE,
    SINGLE_QUOTE,
    DOUBLE_QUOTE,
    SEMI_COLON,
    ENTER,
    SPACE,
    OTHER
} CodeTokenType;
```

每一行为一个双向链表，链表元素是 Token，Line 的定义为：

```
typedef struct {
    LinkedList lineList;
    int length;
} Line;
```

每一个代码文件为一个 Passage，它也是一个双向链表，元素是 Line.这个结构体同时还存储了相应代码文件所对应的字典树（用于匹配前缀，自动补全）

```
typedef struct {
    LinkedList passList;
    Trie trie;
} Passage;
```

这样我们就建立了一个嵌套的链表结构，用来存储代码文件的内容。

附：这里使用的双向链表是我们自己实现的，相关定义如下，不做赘述：

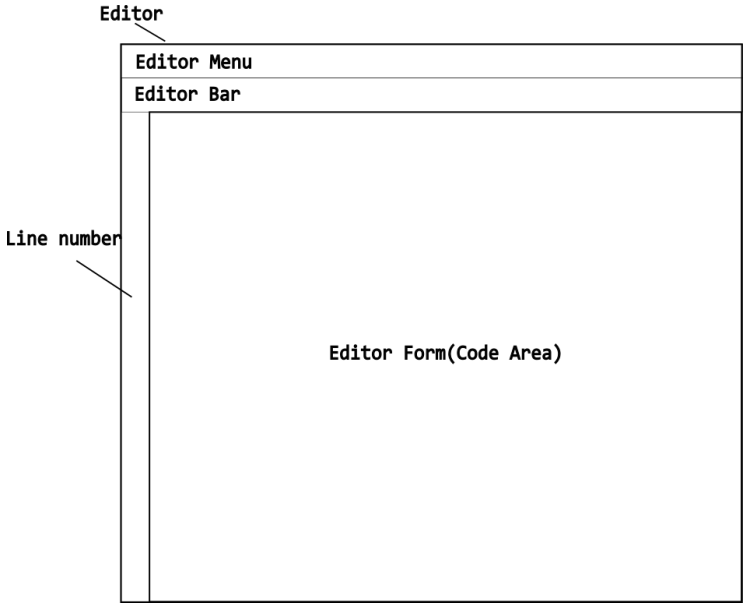
```
typedef struct ListNode {
    void *datptr;
    struct listNode *prev;
    struct listNode *next;
} ListNode;
```

```
typedef ListNode *Listptr;

typedef struct LinkedList {
    Listptr head;
    Listptr tail;
    int listLen;
} LinkedList;
```

字典树定义如下，实现了增、删、改、查的功能：

```
typedef struct {
    struct treeNode *child[52];
    int cnt[52];
    int childNum;
} TreeNode;

typedef struct {
    TreeNode *root;
} Trie;
```

2.下面介绍窗体的数据结构



整个窗体由一个结构体 Editor 保存相应数据，定义如下：

```
typedef struct {
    int fileCount, curSelect;
    double menuHeight, barHeight;
    char *filePath[MAX_FILE_COUNT],
         *fileName[MAX_FILE_COUNT];
    EditorForm *forms[MAX_FILE_COUNT];
```

```
} Editor;
```

该结构体保存了窗体内高度的数据，窗体状态。由于我们的程序支持打开多个文件，所以该结构体还记录了打开文件文件数量，文件名，和内部窗体指针等信息。

当前文件内容在 EditorForm 中展示，相应的定义如下：

```
typedef struct {
    int style, startLine, completeMode;
    bool visible, inSelectionMode;
    double x, y, w, h, viewProgress;
    PosRC caretPos, realCaretPos, renderPos, selectLeftPos,
selectRightPos;
    UndoRedo *urStack;
    Passage *passage;
} EditorForm;
```

记录了字体风格，必要的状态，光标位置信息，以及上文所提到的代码内容结构体 Passage 的指针，和后文要介绍的 UndoRedo 指针。

对于撤消重做，我们通过双向链表来实现，每个节点记录了一次操作，节点数据定义如下：

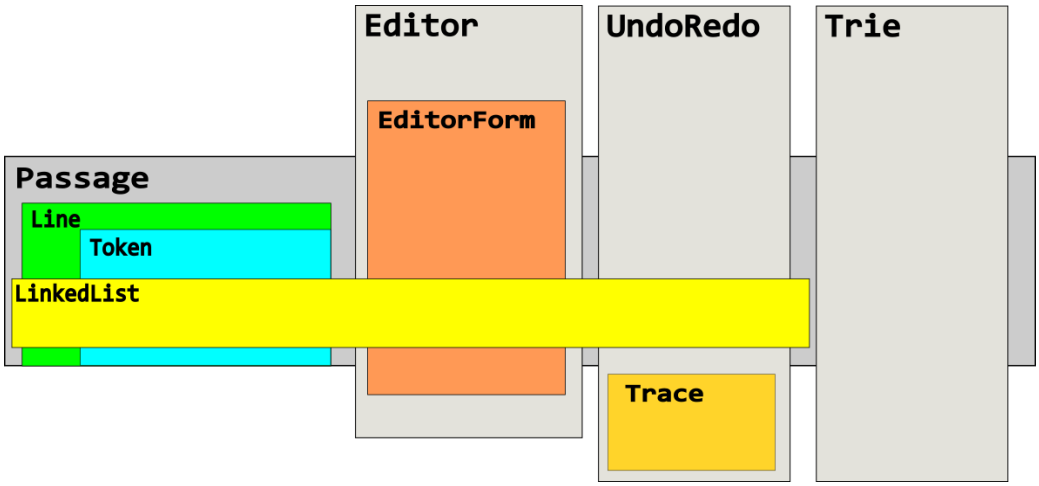
```
typedef enum {
    ADD,
    DELE
} TraceType;

typedef struct {
    TraceType type;
    int rows, cols, rowt, colt; // closed interval
    char content[MAX_LINE_SIZE];
} Trace;
```

由于所有操作都可归结为对代码文件的添加和删除这两种元操作，所以操作类型为添加和删除，并同时记录相应位置和内容。
撤消重做的结构定义如下：

```
typedef struct {
    Passage *passage;
    LinkedList undoRedoList;
    Listptr nowNode;
} UndoRedo;
```

包含操作的代码内容指针，一个双向链表，和当前时间点指针

各个数据结构的嵌套关系如下图所示，**大量运用了链表结构：**



3.4 函数设计描述

```
void initPassage(Passage *p);
```

函数功能	初始化文章
函数参数	Passage *p：需要初始化的指针
函数算法	
所属模块	<codeparser.h> 数据处理模块

```
int parseLine(Passage *passage, int row);
```

函数功能	对指定行进行词法分析
函数参数	Passage *passage：目标文章 int row：行号
重要局部变量	levelCounter 记录括号层级用于自动缩进与括号分级设色 idx 当前扫描位置
返回值	词法分析结束后光标所在的最新行
函数算法	线性扫描，状态记录，链表操作
所属模块	<codeparser.h> 数据处理模块

```
PosRC addString(Passage *passage, char *str, int row, int col);
```

函数功能	在指定位置插入字符串
函数参数	Passage *passage：目标文章 char *str：被插入字符串 int row：插入位置所在行 int col：插入位置所在列
重要局部变量	posRc 光标位置，用于计算添加内容后的新坐标

返回值	新的光标位置
函数算法	链表操作，字符串
所属模块	<codeparser.h> 数据处理模块

```
PosRC deleteString(Passage *passage, int rows, int cols, int rowt, int colt);
```

函数功能	删除指定闭区间的字符串
函数参数	Passage *passage：目标文章 int rows: 区间起始行号 int cols: 区间起始列号 int rowt: 区间终止行号 int colt: 区间终止列号
返回值	新的光标位置
函数算法	链表操作，字符串
所属模块	<codeparser.h> 数据处理模块

```
void initTrie(Trie *tree);
```

函数功能	初始化字典树，将 C 语言关键字存储进字典树
函数参数	Trie *tree：字典树指针
返回值	
函数算法	树操作
所属模块	<trie.h> 数据处理模块

```
void addStringToTrie(TreeNode *root, char *str);
```

函数功能	向字典树添加字符串
函数参数	Trie *tree：字典树指针 Char *str: 待插入字符串
重要局部变量	index 字符所对应的下标，用于寻找子节点
返回值	
函数算法	递归算法，树的深度优先遍历
所属模块	<trie.h> 数据处理模块

```
int deleteStringInTrie(TreeNode *root, char *str);
```

函数功能	删除字典树中指定字符串
函数参数	Trie *tree：字典树指针

	Char *str：待删除字符串
重要局部变量	index 字符所对应的下标，用于寻找子节点
返回值	当前节点类型（根据是否为叶子节点，字符串是否终止判断）
函数算法	递归算法，树的深度优先遍历
所属模块	<trie.h> 数据处理模块

```
TextList *matchPrefix(TreeNode *root, char *str);
```

函数功能	根据前缀字典树中匹配字符串
函数参数	Trie *tree：字典树指针 Char *str：待匹配字符串
重要局部变量	textList 用于保存匹配到的所有字符串
返回值	保存所有合法字符串的链表指针
函数算法	递归算法，树的深度优先遍历，链表操作
所属模块	<trie.h> 数据处理模块

```
PosRC Undo(UndoRedo *ur);
```

函数功能	撤销上一个操作
函数参数	UndoRedo *ur：撤销重做链表
重要局部变量	trace 操作记录
返回值	撤销后新的光标位置
函数算法	链表查找与回溯
所属模块	<undoredo.h> 数据处理模块

```
PosRC Redo(UndoRedo *ur);
```

函数功能	重做上一个操作
函数参数	UndoRedo *ur：撤销重做链表
重要局部变量	trace 操纵记录
返回值	重做后新的光标位置
函数算法	链表查找与遍历
所属模块	<undoredo.h> 数据处理模块

```
void addTrace(UndoRedo *ur, TraceType type, int rows, int cols,
              int rowt, int colt, char *content);
```

函数功能	添加操作记录
函数参数	UndoRedo *ur：撤消重做链表

	TraceType type：记录类型 int rows：起始行号 int cols：起始列号 int rowt：终止行号 int colt：终止列号 char *content：操作字符串
重要局部变量	操作记录
函数算法	链表查找节点、添加节点、删除节点
所属模块	<undoredo.h> 数据处理模块

```
void newFile(Editor *editor);
```

函数功能	新建文件
函数参数	Editor *editor：编辑器结构体
函数算法	字符串匹配，文件创建
所属模块	<ioutils.h> 输入输出模块

```
void loadFile(Editor *editor);
```

函数功能	读取文件
函数参数	Editor *editor：编辑器结构体
重要局部变量	ofn 一个文件结构体实例，存储文件打开路径等信息
函数算法	字符串匹配，文件读取
所属模块	<ioutils.h> 输入输出模块

```
void saveAs(Editor *editor);
```

函数功能	文件另存为
函数参数	Editor *editor：编辑器结构体
重要局部变量	ofn 一个文件结构体实例，存储文件打开路径等信息
函数算法	字符串操作，文件写入
所属模块	<ioutils.h> 输入输出模块

```
void saveFile(Editor *editor);
```

函数功能	文件保存
函数参数	Editor *editor：编辑器结构体
重要局部变量	ofn 一个文件结构体实例，存储文件打开路径等信息
函数算法	字符串操作，文件写入

所属模块	<ioutils.h> 输入输出模块

```
static void moveCaret(EditorForm *form, CaretAction action,
                      char *curLine, char *preLine);
```

函数功能	光标移动
函数参数	EditorForm *form: 代码区域结构体 CaretAction action:光标移动类型 char *curLine: 当前行内容 char *preLine：上一行内容
函数算法	条件分支
所属模块	<ioutils.h> 输入输出模块

```
void handleKeyboardEvent(Editor *editor, int key, int event);
```

函数功能	键盘事件处理
函数参数	Editor *editor：编辑器结构体 int key：按键 int event：按键事件类型
函数算法	条件分支
所属模块	<ioutils.h> 输入输出模块

```
void handleInputEvent(Editor *editor, char ch);
```

函数功能	字符输入事件处理
函数参数	Editor *editor：编辑器结构体 char ch: 输入字符
函数算法	条件分支
所属模块	<ioutils.h> 输入输出模块

```
void handleMouseEvent(Editor *editor, int x, int y, int button, int event);
```

函数功能	鼠标事件处理
函数参数	Editor *editor：编辑器结构体 int x:鼠标横坐标（像素） int y:鼠标列坐标（像素） int button: 鼠标按键 int event: 鼠标事件类型
函数算法	条件分支

所属模块	<ioutils.h> 输入输出模块

```
static void drawEditorMenu(Editor *editor);
```

函数功能	绘制菜单栏
函数参数	Editor *editor：编辑器结构体
函数算法	
所属模块	<textarea.h> 图形绘制模块

```
static void drawEditorBar(Editor *editor);
```

函数功能	绘制类 Dev-C++ 的多文件选择栏
函数参数	Editor *editor：编辑器结构体
函数算法	字符串操作
所属模块	<textarea.h> 图形绘制模块

```
static void drawEditorForm(Editor *editor);
```

函数功能	绘制代码区域
函数参数	Editor *editor：编辑器结构体
函数算法	
所属模块	<textarea.h> 图形绘制模块

```
static void drawCodeLine(EditorForm *form, Line *line,
                         double x, double y, double w, double h);
```

函数功能	绘制单行代码
函数参数	EditorForm *form：代码区域结构体 Line *line: 代码单行内容链表 double x: 左下角横坐标 double y: 左下角纵坐标 double w: 宽度 double h: 高度
函数算法	
所属模块	<textarea.h> 图形绘制模块

```
static void drawEditorComplete(Editor *editor);
```

函数功能	绘制自动补全框体

函数参数	Editor *editor：代码区域结构体
函数算法	
所属模块	<textarea.h> 图形绘制模块

```
static void drawEditorSelection(Editor *editor);
```

函数功能	绘制鼠标选区
函数参数	Editor *editor：代码区域结构体
函数算法	
所属模块	<textarea.h> 图形绘制模块

```
static void drawSymbolMatch(Editor *editor);
```

函数功能	绘制括号匹配
函数参数	Editor *editor：代码区域结构体
函数算法	
所属模块	<textarea.h> 图形绘制模块

3.5 源代码文件组织设计

文件目录结构概览（源代码文件已标蓝）

1）文件函数结构

```
.
├── lib
│   ├── ......
│   ├── extgui.c
│   ├── extgui.h
│   └── ......
├── prjdev
│   ├── CodeEditor.dev
│   └── CodeEditor.ico
├── res
│   ├── icon.h
│   ├── icon.ico
│   └── icon.rc
└── src
    ├── clipboard.c
    ├── clipboard.h
    ├── codeparser.c
    ├── codeparser.h
    ├── doublylinkedlist.c
    ├── doublylinkedlist.h
    ├── help.c
    ├── help.h
    ├── ioutils.c
    ├── ioutils.h
    ├── main.c
    ├── menu.c
    ├── menu.h
    ├── style.c
    ├── style.h
    ├── textarea.c
    ├── textarea.h
    ├── trie.c
    ├── trie.h
    ├── undoredo.c
    └── undoredo.h
```

文件目录介绍：

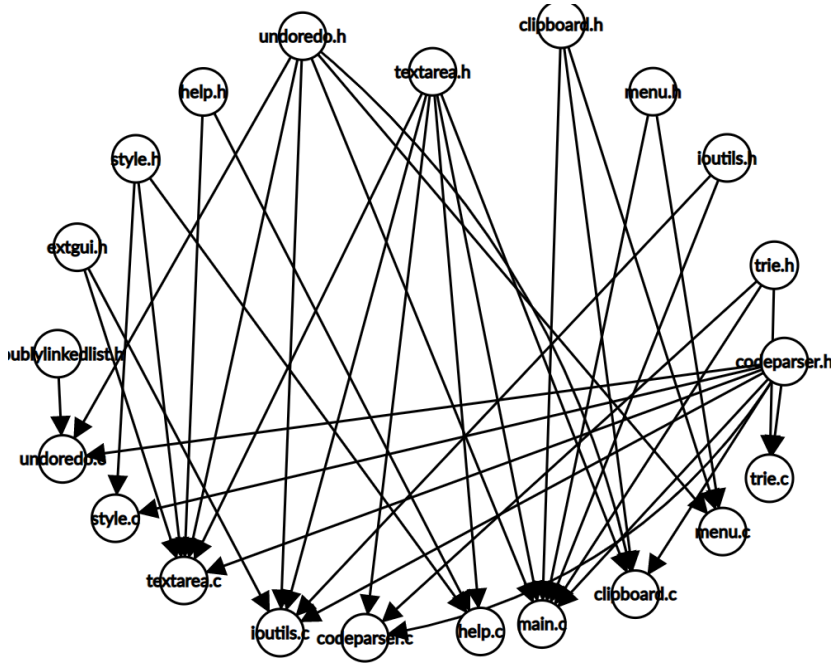
- **lib 文件夹（库文件夹）**
  - libgraphics 库文件
  - imgui 库文件
  - extgui.c, extgui.h 本小组为 imgui 补充的功能代码
- **src 文件夹（项目源代码文件夹，用文件名代表.c 和.h 两个文件）**
  - 数据结构
    - doublylinkedlist 双向链表
    - trie 字典树
  - 输入输出模块
    - clipboard 剪切板读写模块
    - ioutils 输入输出实用函数
    - undoredo 撤销重做模块
  - 代码处理模块
  - codeparser 词法分析模块
  - 图形绘制模块
    - main 主函数
    - menu 鼠标右键菜单
    - help 帮助和关于界面
    - style 主题及配色控制
    - textarea 图形绘制模块
- **res 文件夹（资源文件夹）**
  - icon.ico 轻羽编辑器图标
  - icon.rc, icon.h 资源头文件
- **prjdev 文件夹（Dev-C++项目文件夹）**
  - CodeEditor.dev 项目文件
  - CodeEditor.ico .exe 文件图标

2）多文件构成机制

1. 使用`#define` 头文件保护以防止其被重复包含

```
#ifndef _BLAHBLAH_H_
#define _ BLAHBLAH_H_
// 代码内容
#endif
```

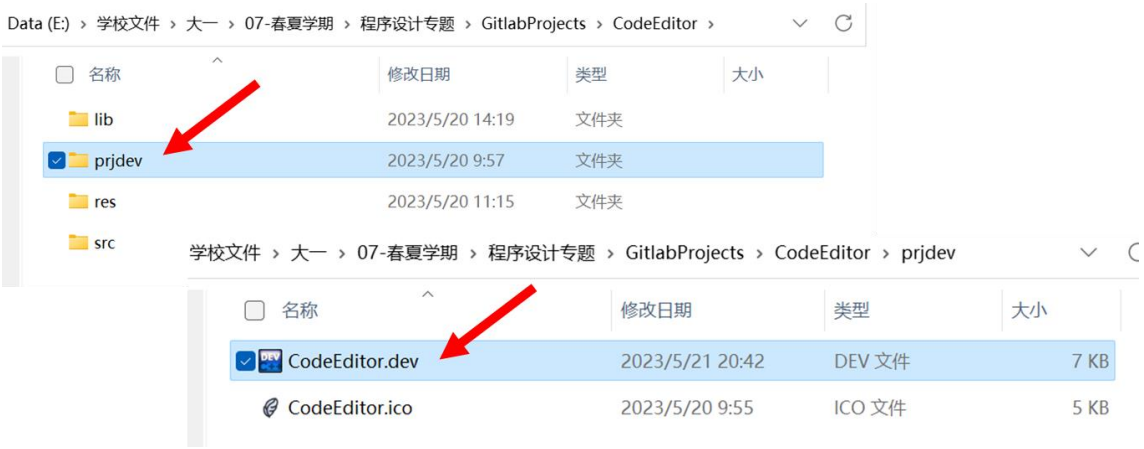
2. 头文件包含关系



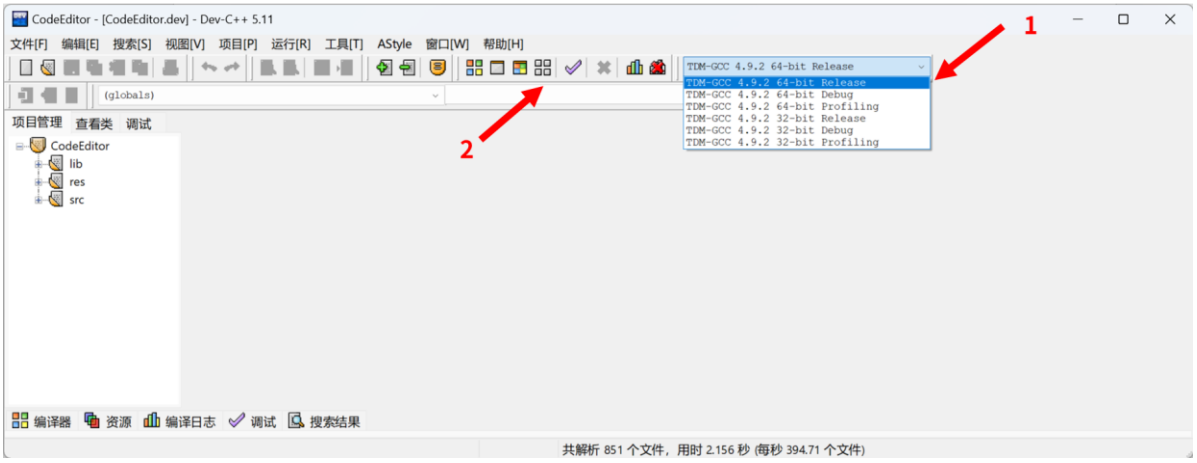
4 部署运行和使用说明

4.1 编译安装

1. 将 zip 压缩包解压后，用 Dev-C++ 打开“prjdev/CodeEditor.dev”



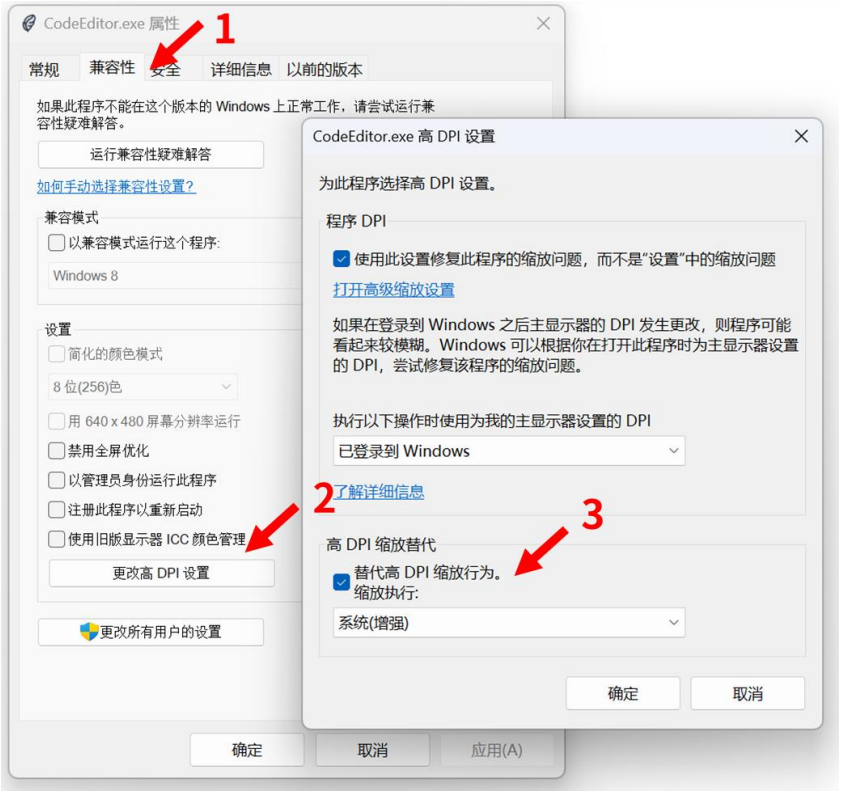
2. 根据电脑系统位数，选择对应的 Release 版本，按 F12 全部重新编译。



3. 编译后，找到根目录下有羽毛笔图标的“CodeEditor.exe”，双击运行

注：如果同学使用的是高分别率屏幕（屏幕分辨率大于 1080p），强烈建议修改 CodeEditor.exe 的高 DPI 设置（也建议同学在互评其他作业时修改）。方法如下：

1. 在 CodeEditor.exe 上右键，单击“属性”
2. 在“兼容性”选项卡中，单击“更改高 DPI 设置”
3. 单选“替代高 DPI 缩放行为”，并在下拉框中选择“系统（增强）”



4.2 运行测试

测试阶段典型案例：括号颜色赋值错误问题

测试数据：一段作业程序中的代码（下图为 Visual Studio Code 截图）

```
#define JUDGE(x, c, s) {\
    FILE *fp = fopen(argv[1], "a");\
    if ( fp ) {\
        fprintf(fp, "%s\t", s);\
        if ( x ) {\
            credit += c;\
            fprintf(fp, "PASS %d %d\n", c, credit);\
        } else {\
            fprintf(fp, "FAIL 0\n");\
        }\
        fclose(fp);\
    }\
}
```

这段代码中的括号在我们的代码编辑器中会出现显示错误，具体表现如下：

```
#define JUDGE(x, c, s) {\
    FILE *fp = fopen(argv[1], "a");\
    if ( fp ) {\
        fprintf(fp, "%s\t", s);\
        if ( x ){\
            credit += c;\
            fprintf(fp, "PASS %d %d\n", c, credit);\
        } else {\
            fprintf(fp, "FAIL 0\n");\
        }\
        fclose(fp);\
    }\
}
```

可以看到，程序默认忽略了第 1 行的预处理指令，而从第 2 行才开始计算括号层级，这就导致括号层级和真正的层级相差 1 级，这也可以解释 12 和 13 行的括号均被绘制为了黄色。

随后我们进行了问题溯源：找到处理括号层级的 codeparser.h，查看其对预处理指令的 case 语句，发现其操作是阻断当前的大循环，直接将该行所有的内容放进 token。这导致了预处理指令中的括号并没有在大循环中被计算，所以层级计算错误。

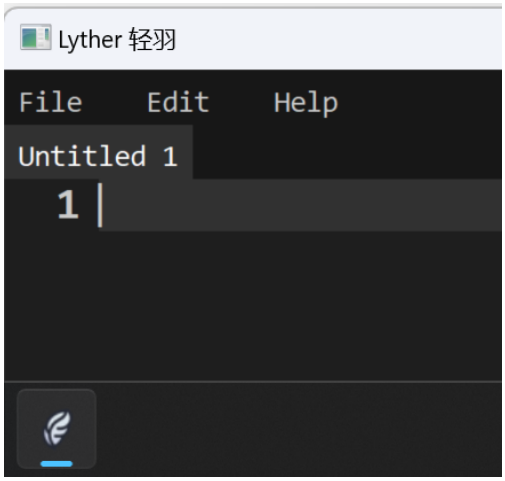
修改：在预处理指令的 case 语句中加入对括号层级匹配的处理。可以发现，修改后括号的颜色显示正常。

```
1 #define JUDGE(x, c, s) {\
2     FILE *fp = fopen(argv[1], "a");\
3     if ( fp ) {\
4         fprintf(fp, "%s\t", s);\
5         if ( x ) {\
6             credit += c;\
7             fprintf(fp, "PASS %d %d\n", c, credit);\
8         } else {\
9             fprintf(fp, "FAIL 0\n");\
10        }\
11        fclose(fp);\
12    }\
13 }
```

4.3 用户使用手册

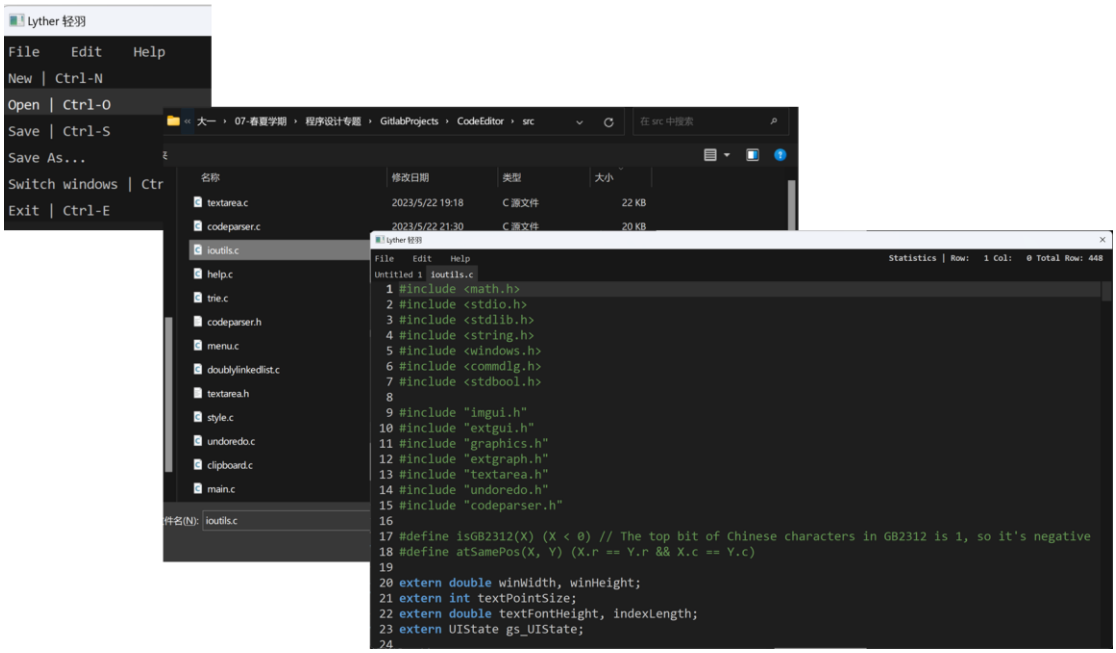
欢迎使用“Lyther 轻羽”代码编辑器！这篇用户使用手册会带你快速上手这款全新的代码编辑器，并且让你进一步了解这款代码编辑器的特色功能。

找到羽毛笔图标的.exe 文件，双击打开“轻羽”编辑器后，会自动新建空白代码文件“Untitled 1”。这样你就可以快速且直接地敲代码啦。

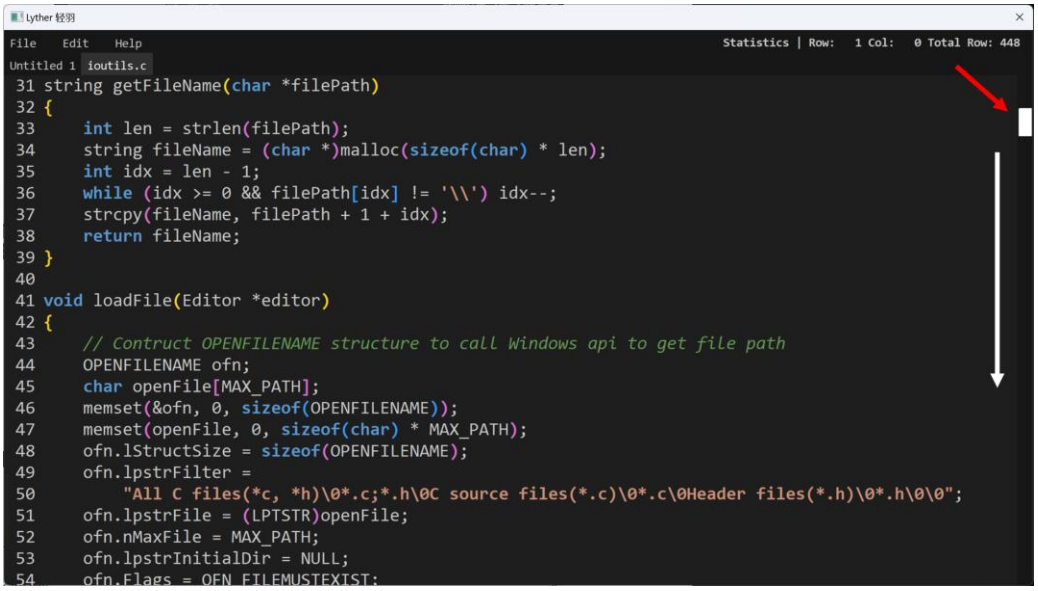


当然，你也可以选择打开本地代码文件，在左上角菜单栏找到“File->Open”，或直接按快捷键“CTRL-O”调出文件选择窗口。

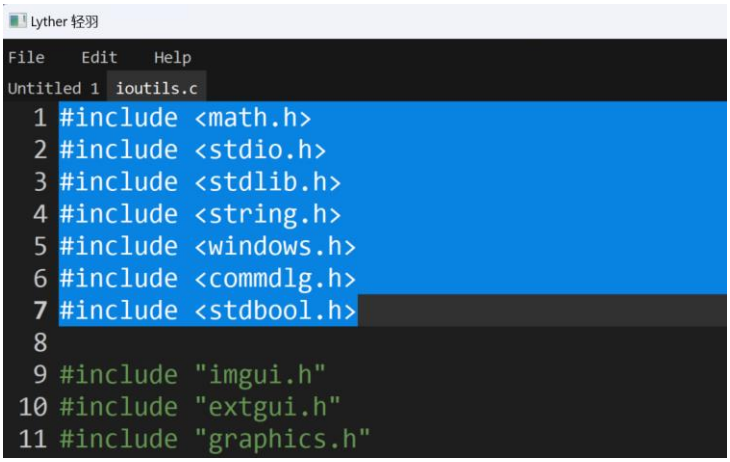
我们以程序源文件中的 ioutils.c 为例。打开代码，你会发现程序为#include, #define 等预处理指令，extern, int, double 等关键字设置了正确的代码高亮格式。同时，对于光标所在的行，我们支持当前行的高亮，以及行号的加粗显示，让你快速找到自己编辑的位置。



如果文件比较长，你可以用鼠标拖拽右侧的滚动条进行代码的快速浏览。

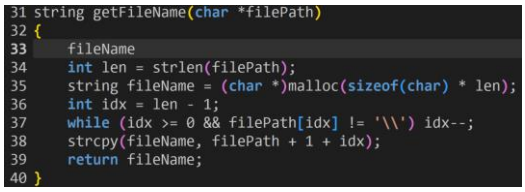


如果想选择一段代码，你可以通过鼠标定位光标，鼠标按住拖动进行选区（与Dev-C++操作类似）。



接下来，如果你想进一步编辑代码文件，我们还为你贴心地准备了两个有力的帮手，它们是——代码补全和自动缩进。

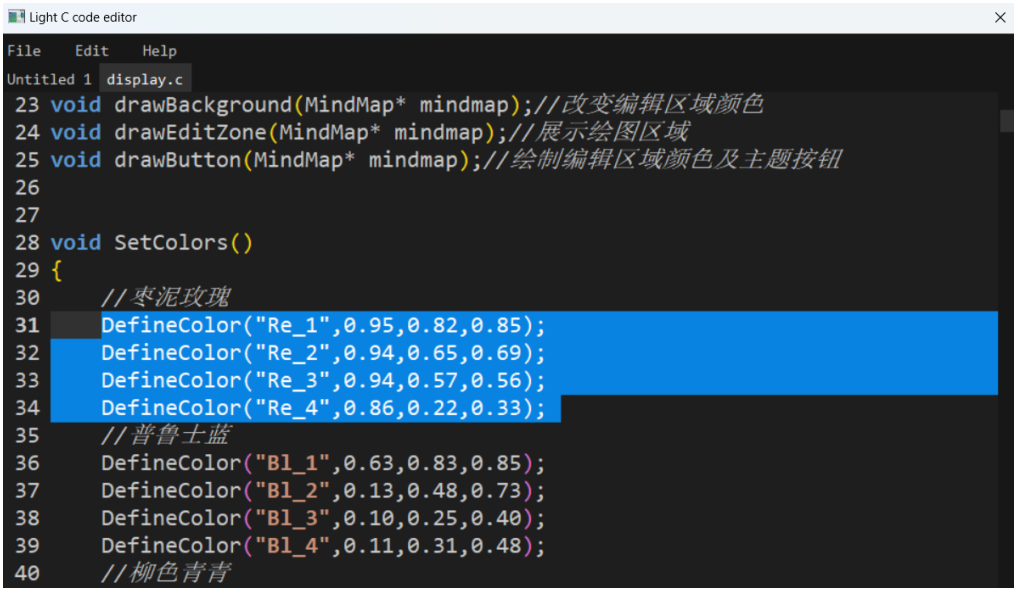
随便打几行代码试试，你就会发现，在光标右下角会出现代码补全的提示框。

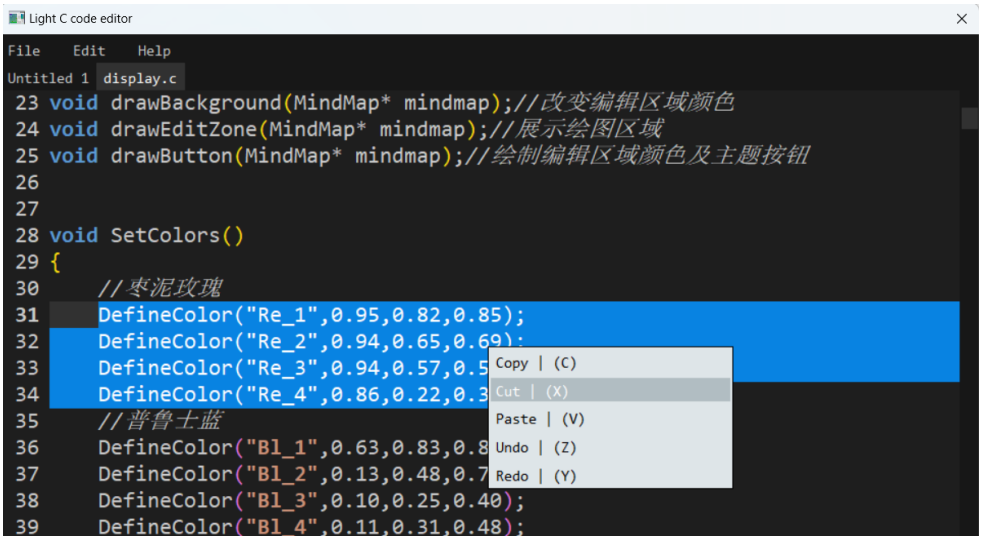


代码补全框显示后，你可以按 ENTER 直接填入第一项，也可以用鼠标选择其它项。

对于常规的编辑快捷键，轻羽编辑器也提供了全面的支持。你可以按 CTRL-Z 来撤销操作，按 CTRL-Y 可以重做，同时在 Edit 菜单或右键菜单也可以进行撤销和重做。

鼠标按住拖动即可选区，CTRL-C, CTRL-X，就可以复制与剪切。这里演示剪切。在示例图中可以看到我们对中文的编辑也是没有问题的。





```
void SetColors()
{
    //枣泥玫瑰
    |
    //普鲁士蓝
    DefineColor("Bl_1",0.63,0.83,0.85);
    DefineColor("Bl_2",0.13,0.48,0.73);
    DefineColor("Bl_3",0.10,0.25,0.40);
    DefineColor("Bl_4",0.11,0.31,0.48);
    //柳色青青
    DefineColor("Gr_1",0.62,0.85,0.78);
    DefineColor("Gr_2",0.53,0.78,0.66);
    DefineColor("Gr_3",0.44,0.79,0.6);
```

而按 CTRL-V 或用相应菜单功能就可以粘贴啦。

Light C code editor

File Edit Help

Untitled 1 display.c display.c

```
19 /*EditPage*/
20 void displayEditPage(MindMap* mindmap);//编辑界面
21 S
22 void drawMenu(MindMap* mindmap); //功能区
23 void drawBackground(MindMap* mindmap);//改变编辑区域颜色
24 void drawEditZone(MindMap* mindmap);//展示绘图区域
25 void drawButton(MindMap* mindmap);//绘制编辑区域颜色及主题按
26
27
28 void SetColors()
29 {DefineColor("Re_1",0.95,0.82,0.85);
30     DefineColor("Re_2",0.94,0.65,0.69);
31     DefineColor("Re_3",0.94,0.57,0.56);
32     DefineColor("Re_4",0.86,0.22,0.33);
33     //枣泥玫瑰
```

编辑完成后，记得按 CTRL-S 保存哦。

5 团队合作

5.1 开发计划

- **前期：**查找资料，了解主流代码编辑器的实现方式；阅读 libgraphics 源码，进一步了解图形库接口。组员 1 和组员 2 共同完成基本数据结构的设计和程序框架的搭建，组员 1 负责初步实现代码处理方面（后端）的基本功能，组员 2 负责初步实现代码编辑器图形化界面（前端）的基本功能。
- **中期：**在前期的基础上，组员 1 进一步完善接口设计，实现目标里基本功能中“编辑”一项的功能及较高级功能，组员 2 通过接口不断完善图形化界面的较高级功能。
- **后期：**仿照 KDE 的“15 分钟 Bug”计划，启动“5 分钟 Bug”计划。组员 1

着力于通过边界和极端条件下的测试，解决程序使用前 5 分钟内遇到的重度影响体验的 Bug。组员 2 编写使用手册和关于界面，调整 UI 设计。

- **末期：** 进入“特性冻结”阶段，不再添加新的功能。组员 1, 2 通过模拟正常使用情况的测试，进一步解决其他小 Bug。最后，添加并整理注释并删除无用的调试信息，对代码格式进行标准化处理。

5.2 编码规范

```
"astyle.cmd_options": [
    "--style=linux",
    // Indent
    "--indent=spaces=4",
    "--attach-closing-while",
    // Padding
    //"--break-blocks",
    "--pad-oper",
    "--pad-header",
    "--pad-comma",
    "--unpad-paren",
    "--unpad-brackets",
    "--delete-empty-lines",
    "--squeeze-lines=2",
    "--squeeze-ws",
    "--align-pointer=name",
    "--align-reference=name",
    "--remove-braces",
    "--keep-one-line-blocks",
    "--convert-tabs",
    "--max-code-length=90",
    "--suffix=none",
    "--verbose",
```

- 本小组使用 Astyle 进行代码规范标准化处理，确保代码风格的明确性和一致性。具体代码规范的要求为:
  - 缩进要求
    - 缩进统一使用空格，而不是\t（libgraphics 不支持显示\t）
    - 缩进长度为 4 格
  - 空格要求
    - 在二元运算符两侧使用空格
    - 在 if, else 等关键字后使用空格
  - 指针描述要求
    - * 统一靠近变量一方
  - 代码长度要求
    - 每行代码长度不超过 90 个字符
- 自定义结构体类型采用大驼峰，其余变量、函数定义采用小驼峰。命名要求清晰展示变量或函数的用途，若名字过长，可省略单词中的元音字母，并做注释。
- 头文件中，每个非 static 自定义类型或函数的声明前，须写注释，包括但不限于如何使用，有什么注意事项。
- 头文件保护，宏名统一为`(_FILENAME_H_)`
- 尽量少使用全局变量

5.3 任务分工

- **组员 1：**
  - 设计存储代码内容的数据结构（上文所述的双向链表嵌套结构），并提供相关函数接口
  - 设计词法分析器，分割词块并划分类型，将代码文件转换为 Token 流。
  - 设计撤销重做数据结构并实现撤销重做功能
  - 为 imgui 模块进行补充，实现右键菜单栏的功能
  - 实现自动补全功能
  - 实现鼠标选区功能
  - 处理文件输入输出和剪切板输入输出
  - Debug
- **组员 2：**
  - 设计储存编辑器图形化界面信息的数据结构
  - 设计代码编辑器的 UI 界面

- 为 imgui 模块进行补充，添加了滚动条的功能
- 实现从 Token 流读取代码并显示代码高亮和行号
- 为词法分析器添加 Token 所处层级的处理并实现自动缩进功能
- 实现主题切换功能
- 处理图形化输入，即鼠标、键盘、字符输入
- Debug

5.4 个人遇到的难点与解决方案

5.4.1 组员 1

1） 嵌套链表结构复杂，指针解引用层数多

画出各指针指向关系，梳理获取某变量所经历的嵌套层数。

2）粘贴大段文字时会溢出 token 最大长度，导致程序崩溃

对粘贴内容进行分块处理（blocking），块大小为 token 最大长度，以多节点形式加入链表。

3）指针大量使用，程序经常因为访问非法地址而崩溃

建立良好的指针管理，避免野指针的出现。在关键函数一开始进行 exception 处理，判断参数合法性，及时返回或返回错误信息。

5.4.2 组员 2

1）没有图层的概念，当鼠标经过多层 UI 时响应混乱

首先对不同 UI 控件的位置，宽度和高度进行清晰的划分，尽可能减少重叠的 UI。然后修改 imgui 的 gs_UIState，使其能够从外部引用，并通过 GenUIID() 生成的唯一号和 gs_UIState.clickedItem 明确当前操作的 UI 控件，让其他控件忽略鼠标操作。

2）代码框架定位不方便，每次刷新都要重绘全部代码造成使用卡顿

在这个问题上，我进行了一定的取舍，放弃了通过修改 form->y 来改变当前显示的部分(VsCode 使用此方案)，而是记录当前第一个可见行来确定(Dev-C++ 使用此方案)。这样虽然没有办法“无极调节”文件显示的进度，但是非常便于计算可见的代码区间，让每次刷新只重绘可见的代码，大大提高了运行的流畅性。

3) 光标在移动时行为和常见的代码编辑器不符合

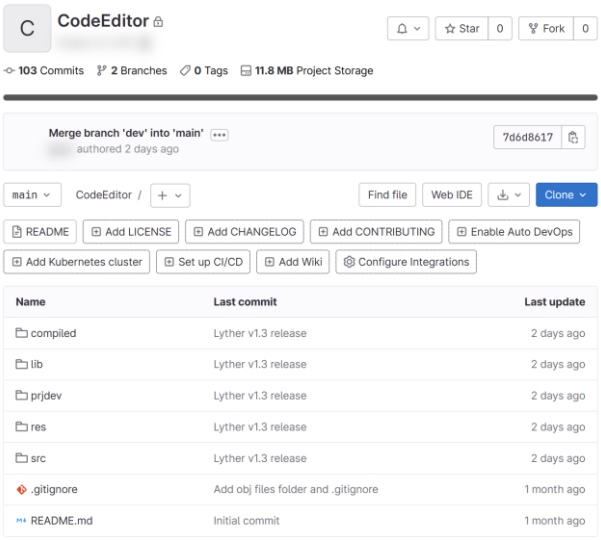
常见的代码编辑器甚至是文本编辑器的光标都十分灵活，比如在行首按左键可以跳到上一行最后，再比如光标向下移动时，经过一个较短的行后还能恢复原来所在的列，而且还能处理中文（双字节字符）和英文混排的情况。我学习了 Dev-C++的解决方案，设计了两套光标位置，一个是光标应该在的位置，另一个是光标的真实位置。同时，我还尽可能地想了光标移动中的不同情况，确保光标在移动中不会跳入中文的中间（双字节字符第一和第二个字符之间），让“轻羽”的使用体验更贴近主流的编辑器。

5.5 工作总结

- 2023 年 4 月 3 日 立项、确定选题并分工

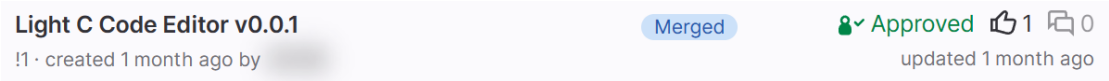
课上公布期末题目候选表格后，小组成员一拍即合决定组队。当晚商议并确定选题为 C 代码编辑器，并大致设计好项目的总体逻辑与代码的各大模块和组员的初步分工。随后组员根据各分工进行自顶向下的开发过程。先对重要数据结

构进行了定义，并实现其基本方法。随后着手开发该项目的关键模块。



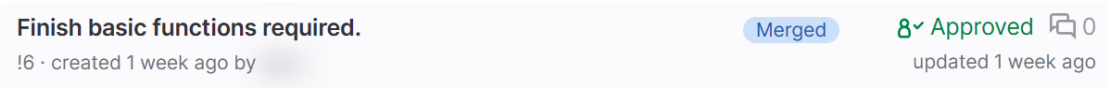
<u>开发亮点：</u>全组在开发全过程均使用 ZJU Git 托管代码，有效解决了组员之间代码同步修改的问题，便于沟通交流和版本控制。项目主要分为 main 和 dev 两个分支，最新修改会被提交到 dev 分支。当实现一定功能并且稳定运行后，在小组成员统一 review 过后，dev 分支将会合并至 main 分支，并更新版本号。

- 2023 年 4 月 4 日 第一次合并——完成了普通文本的简单编辑和显示



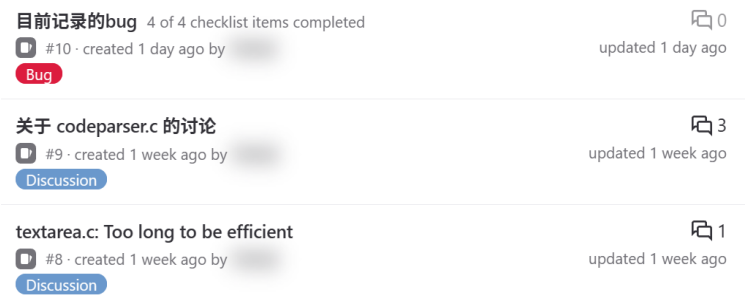
随后，组员进行其它功能的开发设计，例如代码高亮、撤消重做、菜单栏、鼠标选区等。期间也掺杂着 debug 的过程。

- 2023 年 5 月 9 日 第二次合并——完成基本功能要求



在完成了基本功能要求之后（对应开发计划的中期计划），组员继续对界面设计进行完善优化，包括修改 imgui 的主题风格，加入滚动条等。并且进行测试，包括极端条件的程序稳定性和正常使用情况下的测试。

<u>开发亮点：</u>小组成员之间的有效沟通

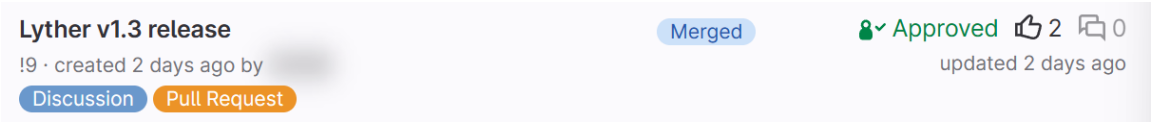


得益于 ZJU Git 平台，小组成员可以通过 Issues 板块进行有关代码功能和 bug 上面的讨论，并且可以直接引用 commit 记录和问题代码，问题修改进度也一目了然，这大大提升了沟通的效率。

➢ 2023 年 5 月 20 日 项目进入“特性冻结”阶段

小组成员不再向代码中添加新的功能，而是着力于 debug。最后，组员 1,2 添加并整理注释并删除无用的调试信息，对代码格式进行标准化处理。

➢ 2023 年 5 月 22 日 项目代码完工



整个项目经历了接近 200 次提交修改，历时一个半月，小组成员所编写的总代码行数超过 2000 行。

5.6 收获感言

➢ 组员 1：

这次大作业主要有以下收获：

1. 熟悉了 git 的使用，可以对代码进行版本控制。
2. 熟悉了一个大项目的开发流程。包括初期的讨论、构想，到中期的实现，到后期的调整。
3. 在制作词法分析器过程中，参阅了 CSAPP 书籍和网络资料，对编译过程有了更多了解。
4. 对于图形程序如何响应事件，及相关信号传递，图形绘制，用户交互等方面有了更多了解。
5. 对于多文件程序的组织形式，项目的目录结构有了更好的认识。
6. 更加熟悉团队交流与讨论

自我评价：我在这次项目中完成了代码内容处理的部分，包括相关数据结构设计、词法分析、撤消重做等。我在项目开发过程中能及时跟进进度，注意队友提出的 issue，并积极进行讨论分析。我还锻炼了查阅资料的能力，比如查看微软的官方文档、计算机书籍等。我也从队友身上学到很多，比如.gitignore 文件的编写，一些命名规范，积极阅读图形库源码并进行模仿扩展等。我这次最大的遗憾就是，限于时间，词法分析的功能做的较为简单，没能深入研究。我的不足是有时候固执己见，不进行商讨就按自己的想法行事。

➢ 组员 2：

体会心得 & 经验教训：

在写图形化界面的前期，我没有太注意元素的定位方式，过多地使用了绝对坐标的定位，而忽略了根据代码框体的坐标，宽度和高度计算出来的相对位置，这导致了后面写滚动条的时候比较麻烦，也不便于增加新的功能，比如分屏显示等等。如果能够回到项目开始的那天，我会让程序设置好每个代码框体的坐标和尺寸，并通过坐标、尺寸作为参数计算出各个 UI 组件的相对位置，而非默认以(0,0)为原点。

在程序开发的前期，没有过多地注意到指针的问题，导致我们后期遇到的几乎所有闪退 bug 都是读取了 NULL 指针导致的。如果能够回到项目开始的那天，我会给所有需要用到指针的接口加上判空的处理。在项目结束后，我会去了解一

些不需要烦恼指针问题的编程语言是如何处理指针的。

自我评价：

在这次项目中我主要负责图形化界面的编写部分，也涉及了部分的代码处理模块。第一次接触编程类的大作业，尽管题目极具挑战性，但是我还是坚持了下来，这让我感觉自己真的已经入迷了。多少个周末我会坐在电脑前调一整天代码，把其他作业抛之脑后，多少个凌晨我会打开 ZJU Git，重新检查自己刚刚提交的代码。我会找到 Dev-C++的 Pascal 源码去硬啃，我会在五一假期里点开智云课堂中的《编译原理》课程自学（虽然很难看不懂）……这个项目驱使着我们去学习，再把学习的成果输出到代码中。

春学期的《程序设计专题》课程，加上一个半月的开发历程，真得让我收获了很多，成长了很多。我学会了 git 在多人协作时的基本操作，了解了代码编辑器是怎么高亮代码的，浅浅地窥探了一下编译器的词法分析和语义分析流程，还更深入理解了及时响应的 UI 的原理，以及回调函数的使用（感觉在游戏制作还有前端编程里都会用到这样的模型）。本以为编个小程，结果整了个代码编辑器，我觉得，写了这样的程序，值了！

6 参考文献资料

1. https://en.wikipedia.org/wiki/Trie 字典树

2. *Engineering a Compiler* – By Keith Cooper 编译原理（虎书）