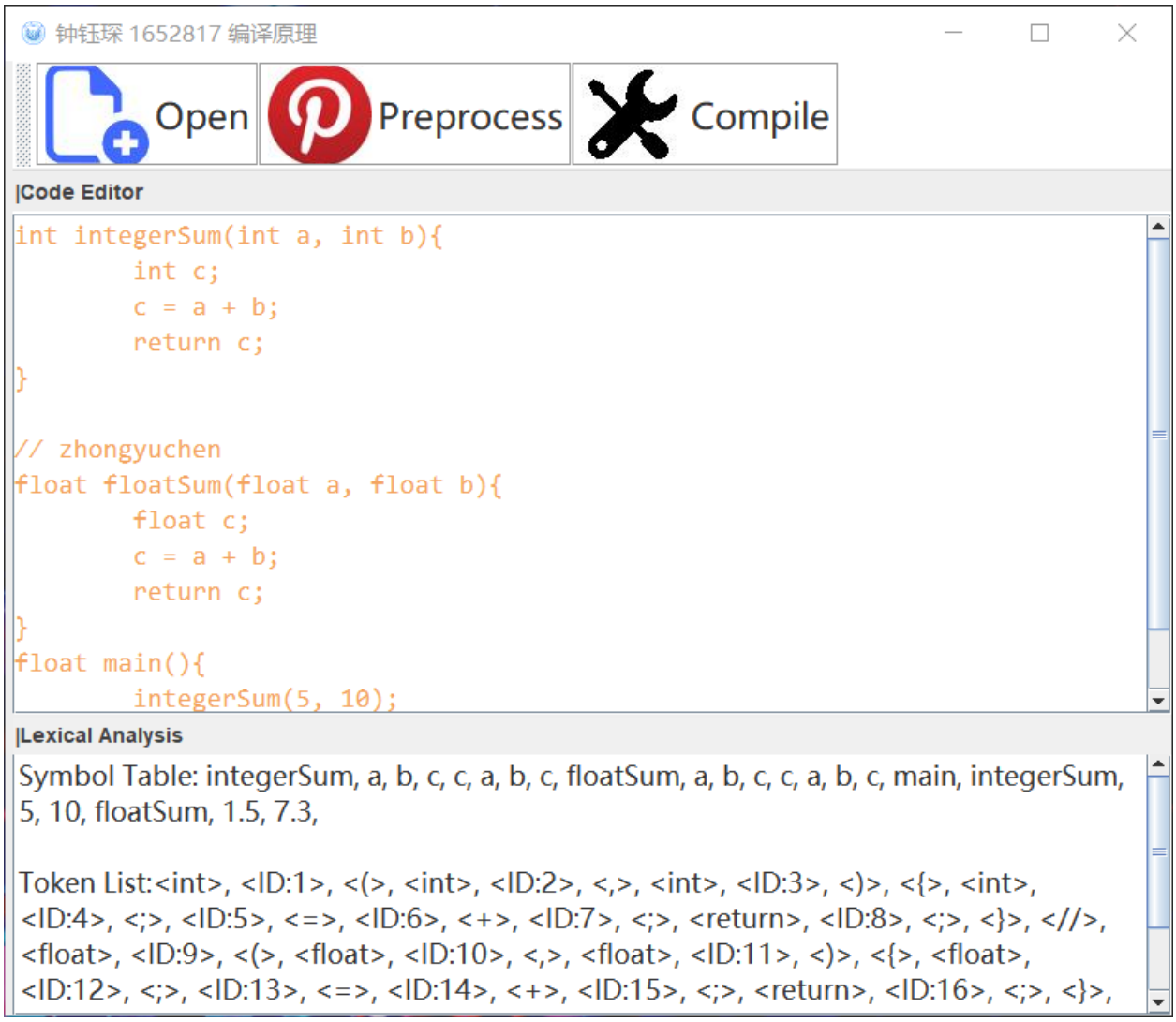
编译原理

C-Like 语言的词法分析器项目文档

1652817 钟钰琛 计算机科学与技术



简介

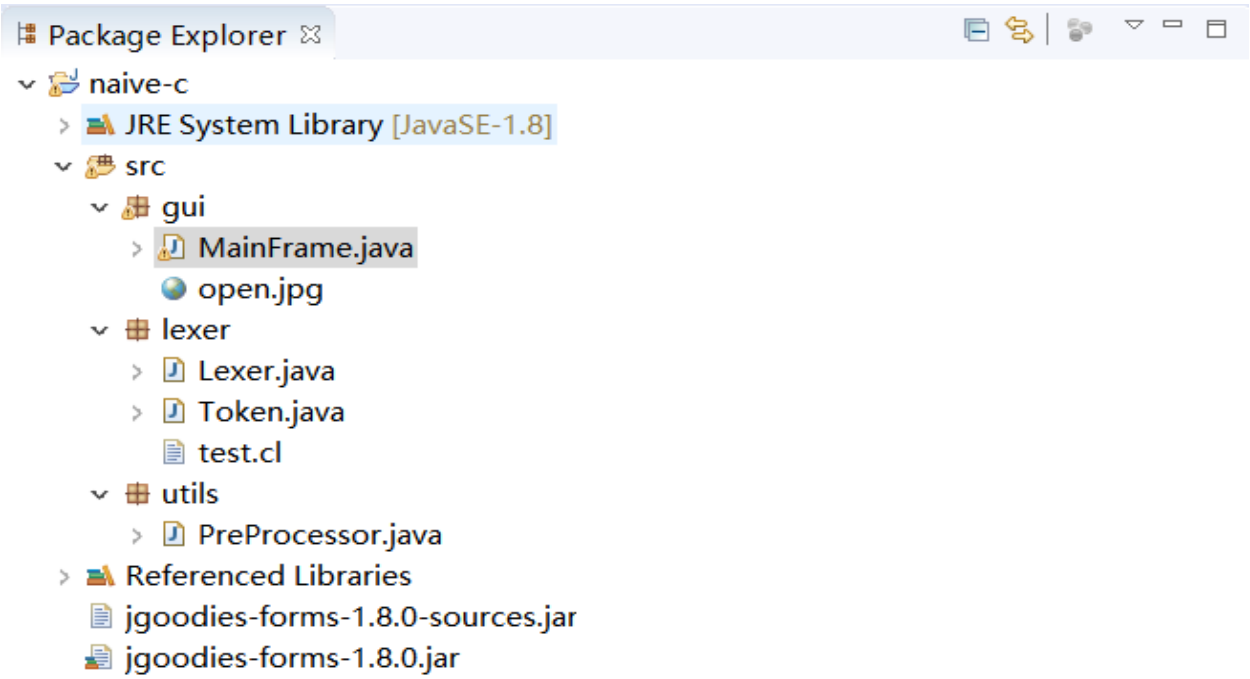
语言：java

JDK: version "1.8.0_171"

开发 IDE： eclipse

GUI 使用 Java Swing 开发

文件目录结构：



lexer 包：实现词法分析器的主要包

Lexer.java: 实现词法分析器

Token.java: 包含 C-like 语言的一些词法规则，以及生成的 <token-name, attribute-value>对

utils 包：utilities，一些常用的模块

PreProcessor.java：封装了预处理模块

gui 包：实现 GUI

MainFrame.java: 带 main 函数执行入口，GUI 的主要定义部分

设计与实现

Token 类：包括 C-Like 语言的词法规则与单词符号对

数据成员：

1.词法规则：

用 ArrayList 来保存 C-Like 语言的词法规则

```
12    // rules of C-like lang
13    private static List<String> keywords = new ArrayList<String>();
14    private static List<String> delimiters = new ArrayList<String>();
15    private static List<String> operators = new ArrayList<String>();
16    private static List<String> demicals = new ArrayList<String>();
17
18    static {
19        // add keywords
20        Token.keywords.add("int");
21        Token.keywords.add("void");
22        Token.keywords.add("if");
23        Token.keywords.add("else");
24        Token.keywords.add("while");
25        Token.keywords.add("return");
26
27        Token.keywords.add("float");
28
29        // add delimiters
30        Token.delimiters.add(";");
31        Token.delimiters.add(",");
32        Token.delimiters.add("(");
33        Token.delimiters.add(")");
34        Token.delimiters.add("{");
35        Token.delimiters.add("}");
36
37        // decimal dot
38        Token.demicals.add(".");
39
40        // add operators
41        Token.operators.add("+");
42        Token.operators.add("-");
43        Token.operators.add("*");
44        Token.operators.add("/");
45        Token.operators.add("=");
46        Token.operators.add("==");
47        Token.operators.add("<");
48        Token.operators.add("<=");
49        Token.operators.add(">");
50        Token.operators.add(">=");
51    }
52
```

这样做的目的是为了方便后续搜索.

（红框圈出来的是浮点数所需要的 float 类型和小数点.）

2. 单词符号：

```
private String token_name;
private Integer attr_value;
```

词法分词器的返回是形如

<token-name, attribute-value>

这样的单词符号对

成员函数:

1. `public static boolean isKeyword(String token);`

判断给定的 token 是否是关键字

2. `public static boolean isDelimiter(String token);`

判断给定的 token 是否是分隔符

3. `public static boolean isOperator(String token);`

判断给定的 token 是否是操作符

（这三个函数的实现是通过在数据成员的对应 List 中直接寻找即可）

4. public String toString();

用来打印 token 的函数

Lexer 类：词法分析器的具体实现

数据成员：

1.常量：

```
static final int NO_VALUE = 0;
```

许多 token 比如 +, - , int, void 等等，没有对应的属性值，所以用这个特定的值来代替.

```
static final int OUT_CMT = 0;
```

状态量，表明词法分析的当前符号不在注释中

```
static final int OUT_CMT = 1;
```

状态量，表明词法分析的当前符号在”//”的注释中

```
static final int OUT_LCMT = 2;
```

状态量，表明词法分析的当前符号在/* */的注释中

2. 符号表(symbol table)

```
private List<String> symbol_table = new ArrayList<String>();
```

用来保存所有出现过的标识符(identifier)与数字(number).

3. 单词符号对表(token list)

```
private List<Token> token_list = new ArrayList<Token>();
```

保存词法分析结果的列表

成员函数：

1.`private boolean isDigit(char ch)`

判断字符是否是数字

2.`private boolean isLetter(char ch)`

判断字符是否是字母（A-Z 或者 a-z）

3.`private boolean isIdentifier(String input)`

判断字符串是否是标识符（由字母和数字组成，首字符必须是字母）

4.`private boolean isNumber(String input)`

判断字符串是否是数字（包括整数和浮点数）

5.`public String lexAnalysis(String input)`

词法分析（核心函数，后面在算法设计处详细介绍） 按行输入

PreProcessor 类：预处理器

无数据成员

成员函数：

1.public boolean CheckExt(String fileName)

检查打开的文件是否合法（必须以.cl 结尾）

2.public String preprocess(String input)

预处理，一行一行读，每行去掉首尾的空白字符，并且删除注释行

算法设计：

词法分析器每次读入一个字符，判断是否是空白或者是制表符，如果是的话就跳过；

如果不是，将该字符加入到候选 token 的尾部，候选 token 是一个字符串。

接着候选 token 是否已经满足了某些条件：

1. 是否是#结束符？如果是那么就添加到 Token 列表中，并且清空候选 token。
2. 是否是//注释？如果是则同上，并且设置状态量为 IN_CMT，表示目前处于//注释当中（如果处在//注释中，直到遇到换行符才离开注释）
3. 是否是/*注释？如果是则同上，但是状态量设置为 IN_LCMT，表示表示目前处于/*注释当中（如果处在//注释中，直到遇到*/才离开注释）
4. 是否是*/注释？如果是则同上，但是状态量设置为 OUT_CMT,因为此时已经离开了注释区域
5. 是否是分隔符？如果是，并且不处于注释区域（状态量为 OUT_CMT），那么就添加到 Token 列表中。清空候选 token。
6. 是否是操作符？这里有些特殊情况，需要下面介绍的超前搜索。第一个是如果下一个符号是“=”，那么说明当前操作符没有结束，因为“>=”, “<=”和”==”都是有两个符号的；第二个是如果当前符号是“/”，下一个符号是”/“或者是“*”，那么说明这个不是除法运算符，而是注释；第三个是如果当前符号是“*”，而下一个字符是“/”，那么说明这个不是乘法运算符，而是注释。如果是这些特殊情况，那么直接 continue；如果不是，并且不处于注释区域（状态量为 OUT_CMT），那么就添加到 Token 列表中。清空候选 token。
7. 是否是关键字？这里也需要超前搜索，虽然现在匹配到了关键字，但是如果下一个字符是字母或者数字，那么说明这个是一个标识符而不是关键字。如

果出现这种情况，也直接 continue；如果不是，并且不处于注释区域（状态量为 OUT_CMT），那么就添加到 Token 列表中。清空候选 token。

8. 是否是标识符？标识符是由字母或者数字组成，并且首字符必须是字母。这里也需要超前搜索，决定标识符是否结束或者包含不合法的字符。
9. 是否是数字？这里包括整数和浮点数。浮点数是有小数点的。所以数字至多有一个小数点。这里也需要超前搜索，决定数字是否结束或者包含不合法的字符。
10. 如果以上都不是，说明出现了不合法的字符。报错！

需要的技巧：

超前搜索：当前字符不足以判断，需要后续字符的值一起来判断该 token 属于什么类别。超前搜索需要判断是否到了行的尾端，因为标识符、数字、关键字、操作符都不允许跨行。

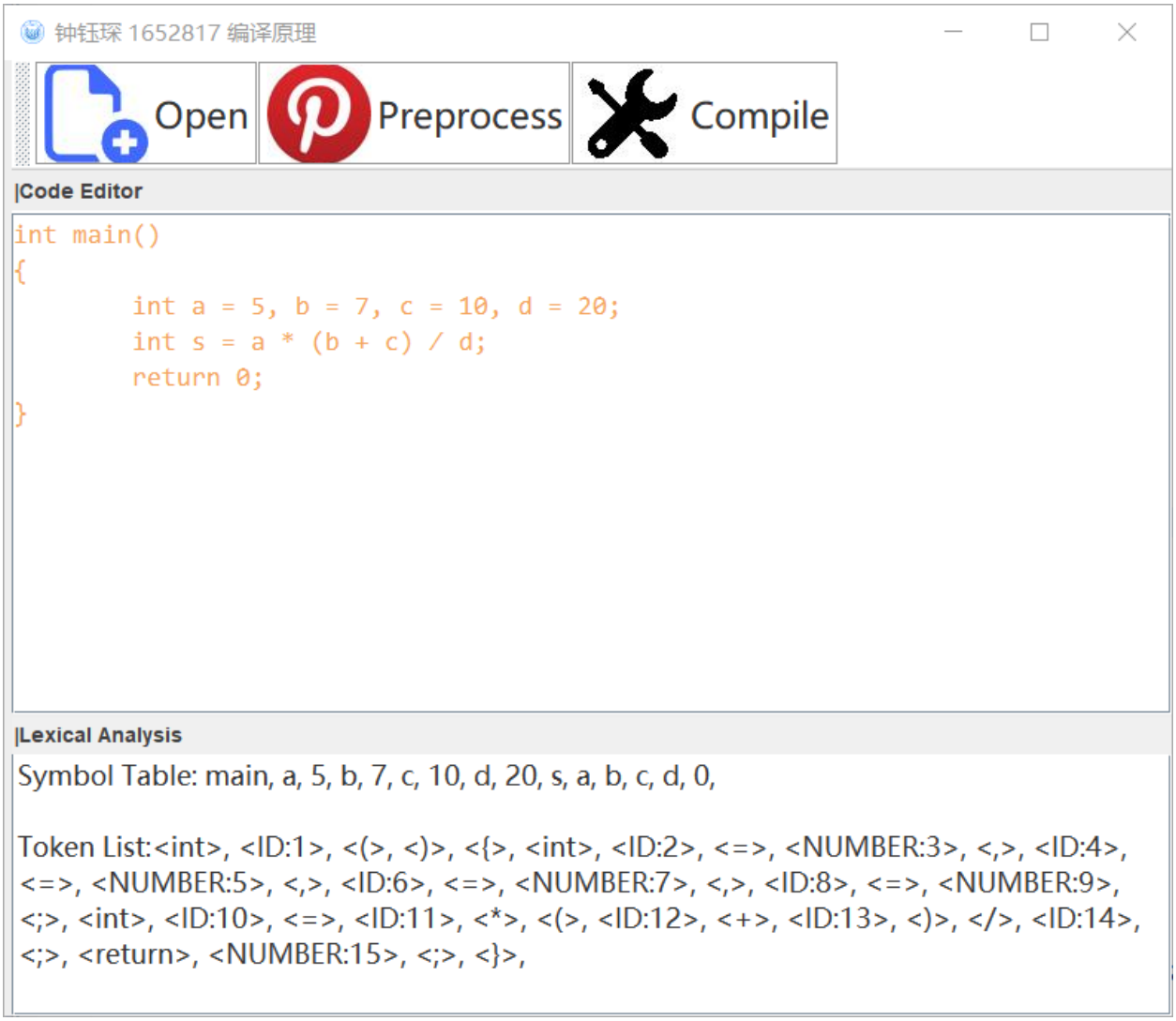
使用说明

1. 打开文件：会判断是否是 xxx.cl 结尾的文件，如果不是，会在代码编辑框显示红色报错信息. 读取成功后，文件的内容会显示在代码编辑框内。
2. 预处理：会对代码编辑框内的内容进行预处理，包括 trim 和删除注释行。
3. 执行词法分析：这里用了 compile 这个词，是为了后续继续做的考虑。结果将显示在下面的文本框中。

4. 代码编辑框：橙色字体。还没弄代码高亮……

5. 词法分析结果：会打印两个，一个是符号表(symbol table)，一个是 token 表

调试分析



符号表有 main a b c d 以及 5 7 10 20 0　全部正确

词法分析表：关键字<int> <return>　分隔符：() {} , ;　运算符: + * /

标识符：<ID:1> 对应 main，<ID:2>对应 a, <ID:4>对应 b，<ID:6>对应 c, <ID:8>对应 d

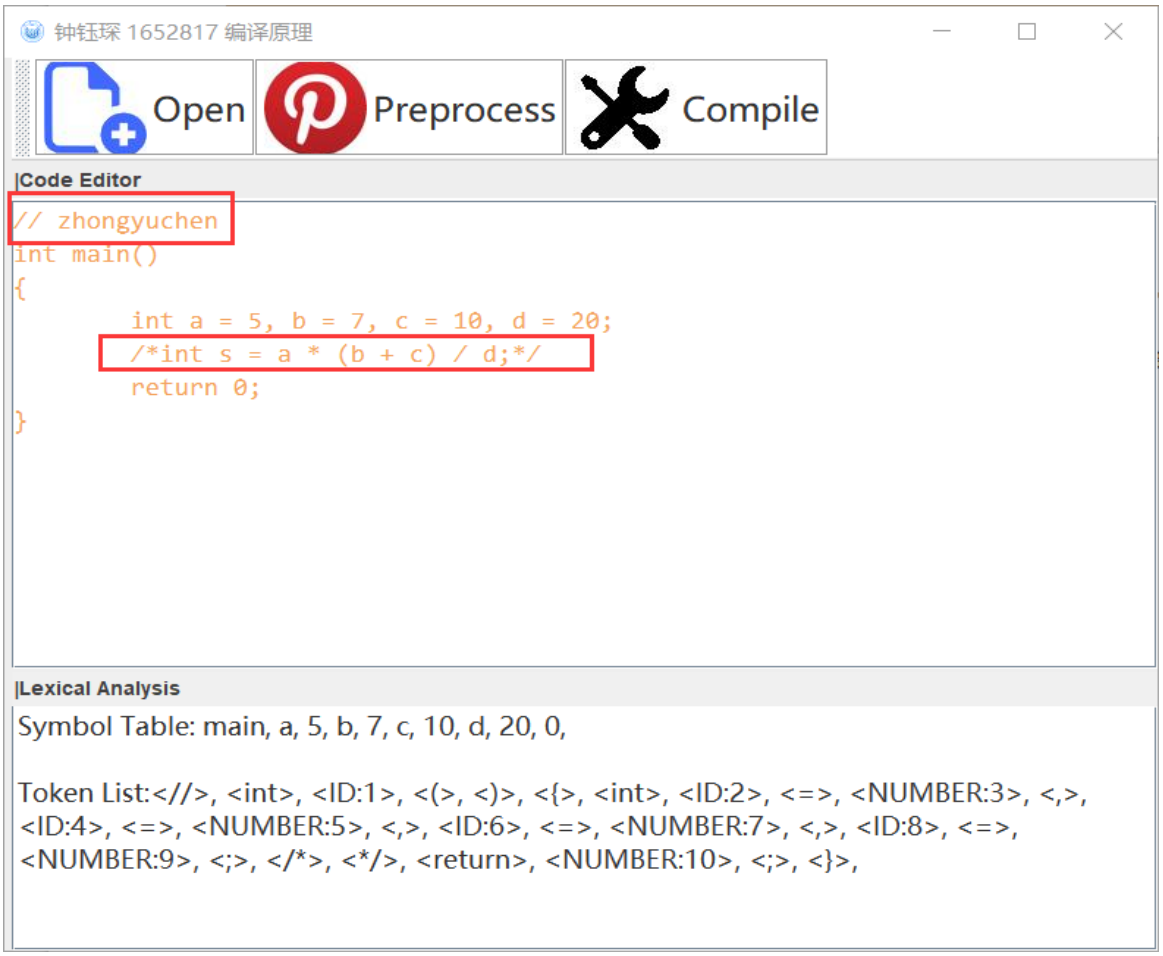
<ID:10>对应 s，<ID:11>、<ID:12>、<ID:13>、<ID:14>对应 a b c d

数字：<NUMBER:2>对应 5 <NUMBER:5>对应 7, <NUMBER:7>对应 10, <NUMBER:9>对应 15

<NUMBER:15>对应 0

所以全部正确.

在此基础上添加两个注释（已用红框标出）



此时可见，在符号表中，没有了 s

在词法分析表中，多了//、 /*和 */，但是没有+、 *和 /

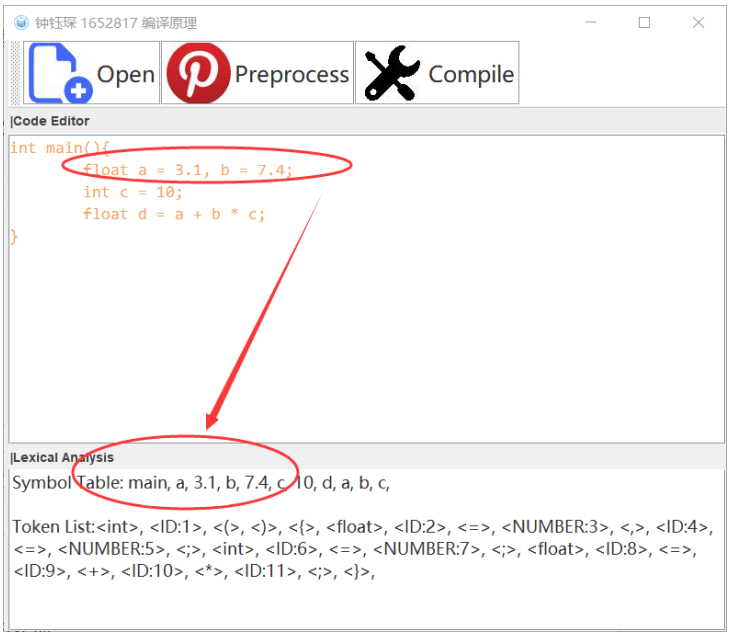
程序正确

BONUS

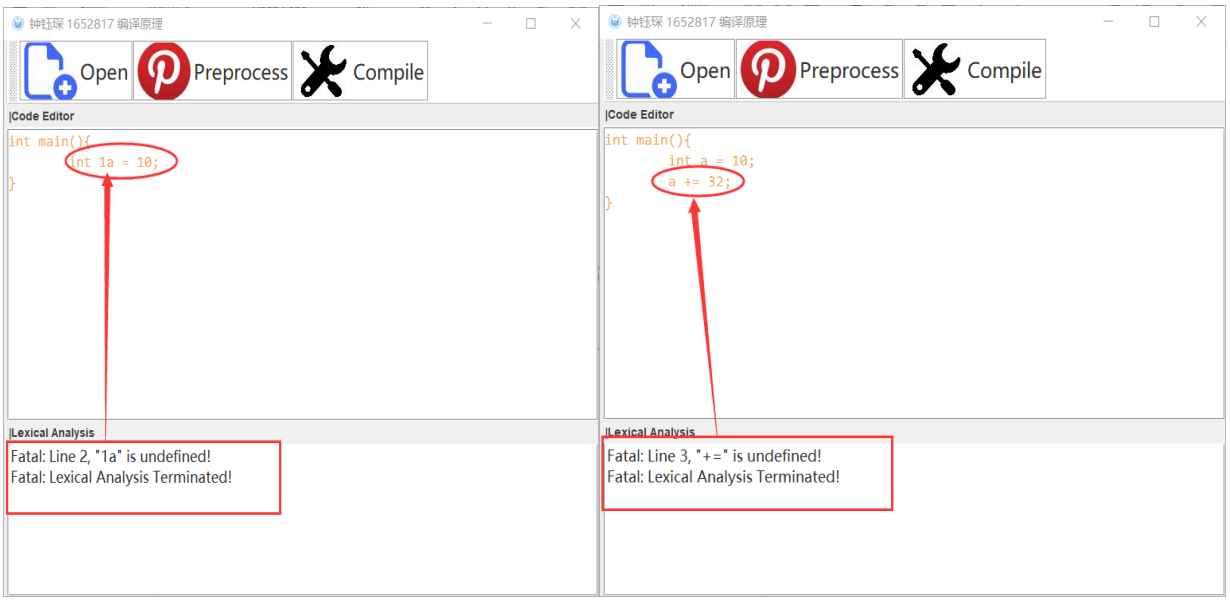
1.添加单词数量。在本程序中修改单词数量极为简单。只需要在 Token.java 里面直接添加即可。

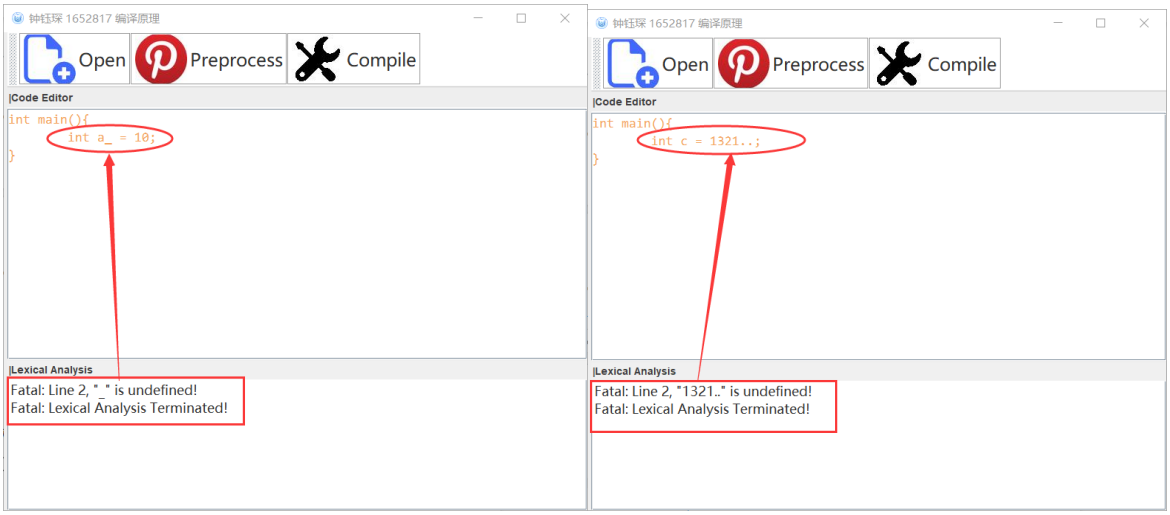
```
Token.keywords.add("float");
```

2. 将整常数扩充为实常数。已实现。调试结果如下：



3. 增加出错处理功能。已实现。调试结果如下:



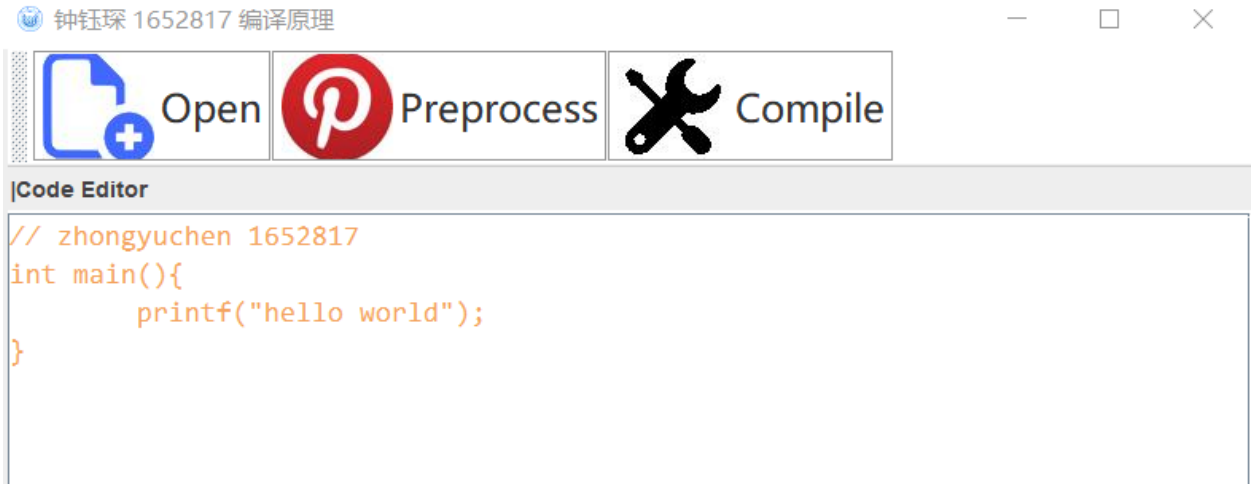


这些都是不符合词法规则的词。

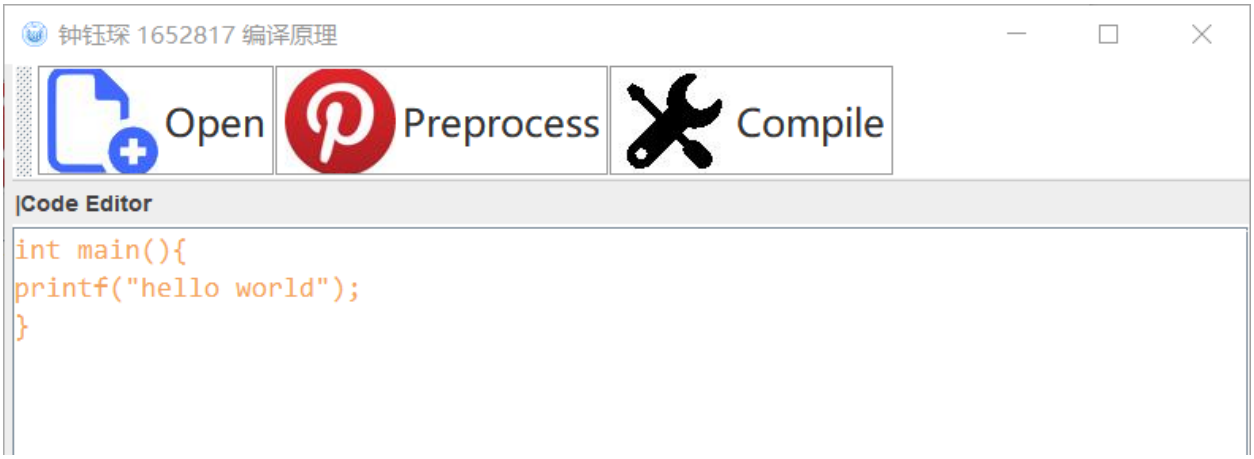
4. 增加预处理程序，每次调用时都将下一个完整语句读入扫描缓冲区，去掉注解行，并能对源程序列表打印。已实现。

预处理器每次将去除每行的头尾空格，如果是//行则直接取出，如图所示

预处理前



预处理后



源程序列表打印，这个不方便放入 gui 所以我就输出到了命令行里面。具体是把递归遍历文件夹然后扫描所有 xxx.cl。

```
test
    1
        1.cl
        2.cl
    2
        1.cl
        2.cl
    3
        1.cl
        2.cl
```

```java
public static void sourceCodeList(File file, int i) {
    for(int j = 0;j < i * 5; ++j) {
        System.out.print(" ");
    }

    if(file.isDirectory()) {
        System.out.println(file.getName());
        File[] files = file.listFiles();
        for(File f : files) {
            sourceCodeList(f,i + 1);
        }
    }else {
        if(CheckExt(file.getName())) {
            System.out.println(file.getName());
        }
    }
}
```

项目开源地址：https://github.com/zhongyuchen/naive-c

参考资料：

[1]: Making Text Editor with Java Swing Tutorial #1

https://www.youtube.com/watch?v=yzc5-JlN-Yc

[2]: Dragon Book ch3 lexical analysis