# 实验内容

完成 PythonT 语言词法分析器的设计，包括lex实现与C++实现。

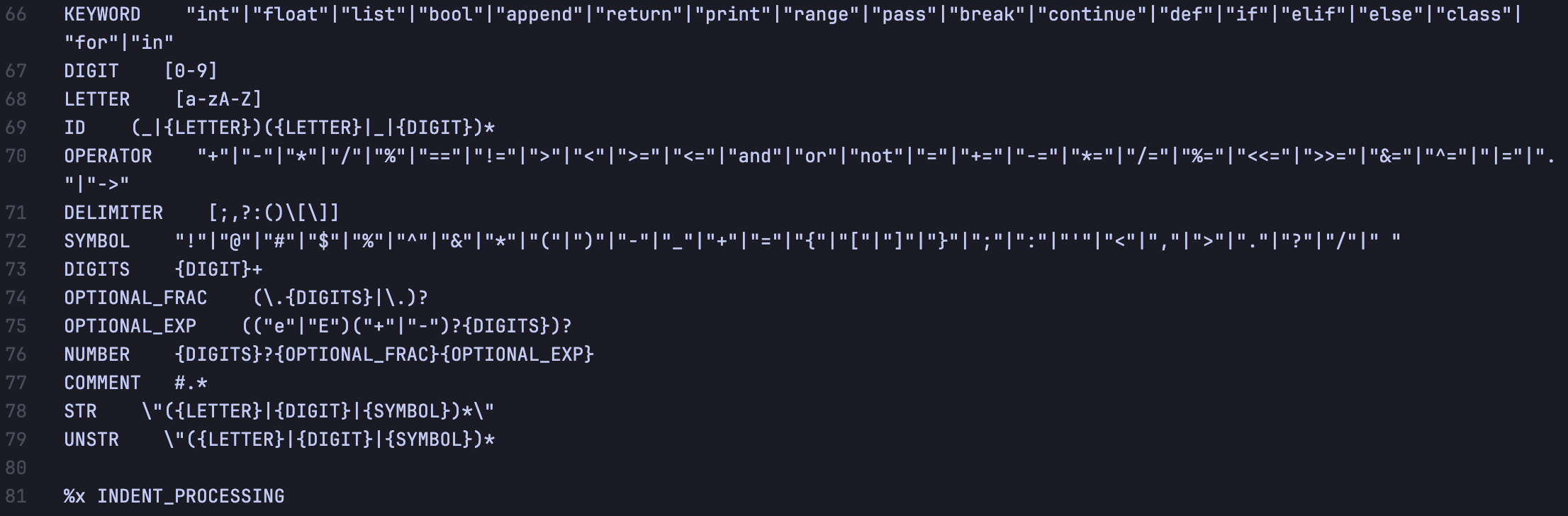
# Lex实现

## 初始化

图 2‑1 初始化代码

初始化部分，引入标准库与计数器，创建缩进栈结构体，定义栈操作函数与缩进处理逻辑。定义两个特殊全局变量，line\_num与full\_comment，line\_num用于指示逻辑行数，full\_comment作用在2.3有详细说明。

## 规则预定义

图 2‑2 规则预定义代码

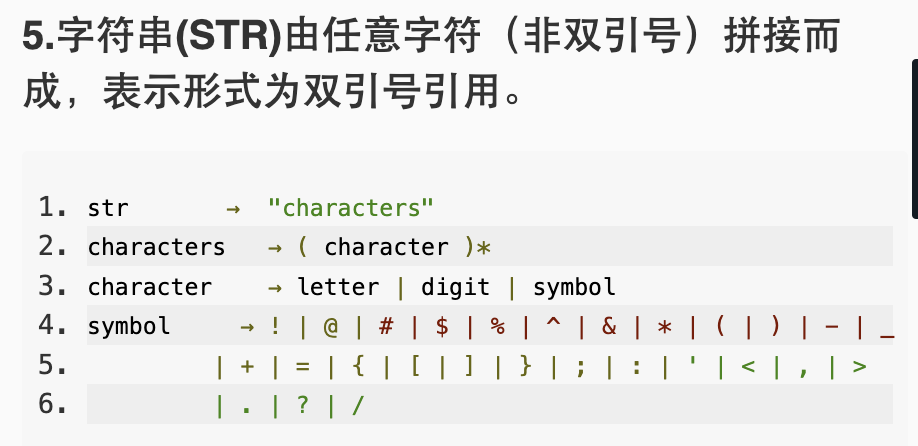
为了避免影响代码可读性，按照PythonT各类单词组成进行规则预定义，其中包括关键字（KEYWORD）、标识符（IDENTIFIER）、运算符（OPERATOR）、分隔符（DELIMITER）、字符串（STRING）、数值常量（NUMBER）以及各种基本组成元素，此外将INDENT\_PROCESSING设置为独占模式，即一旦进入INDENT\_PROCESSING类型的规则后，非INDENT\_PROCESSING类型的规则自动屏蔽。

需要注意的是，**实际测试案例中有多处与实验平台所述规则存在出入**：

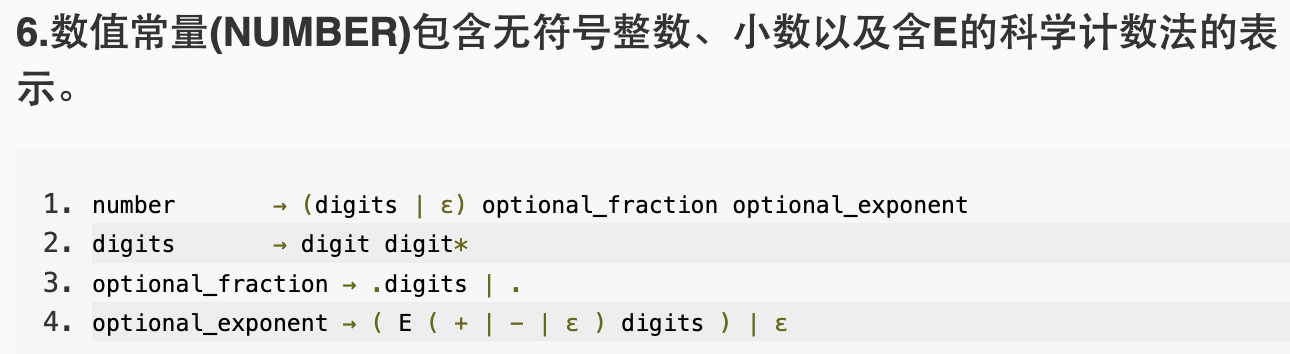
1. True和False在规则中被归类为关键字，实际测试案例中按照标识符处理；

图 2‑3 将True归类为标识符示例

1. 字符串产生式中，非终结符symbol所能推出的字符缺少空格；

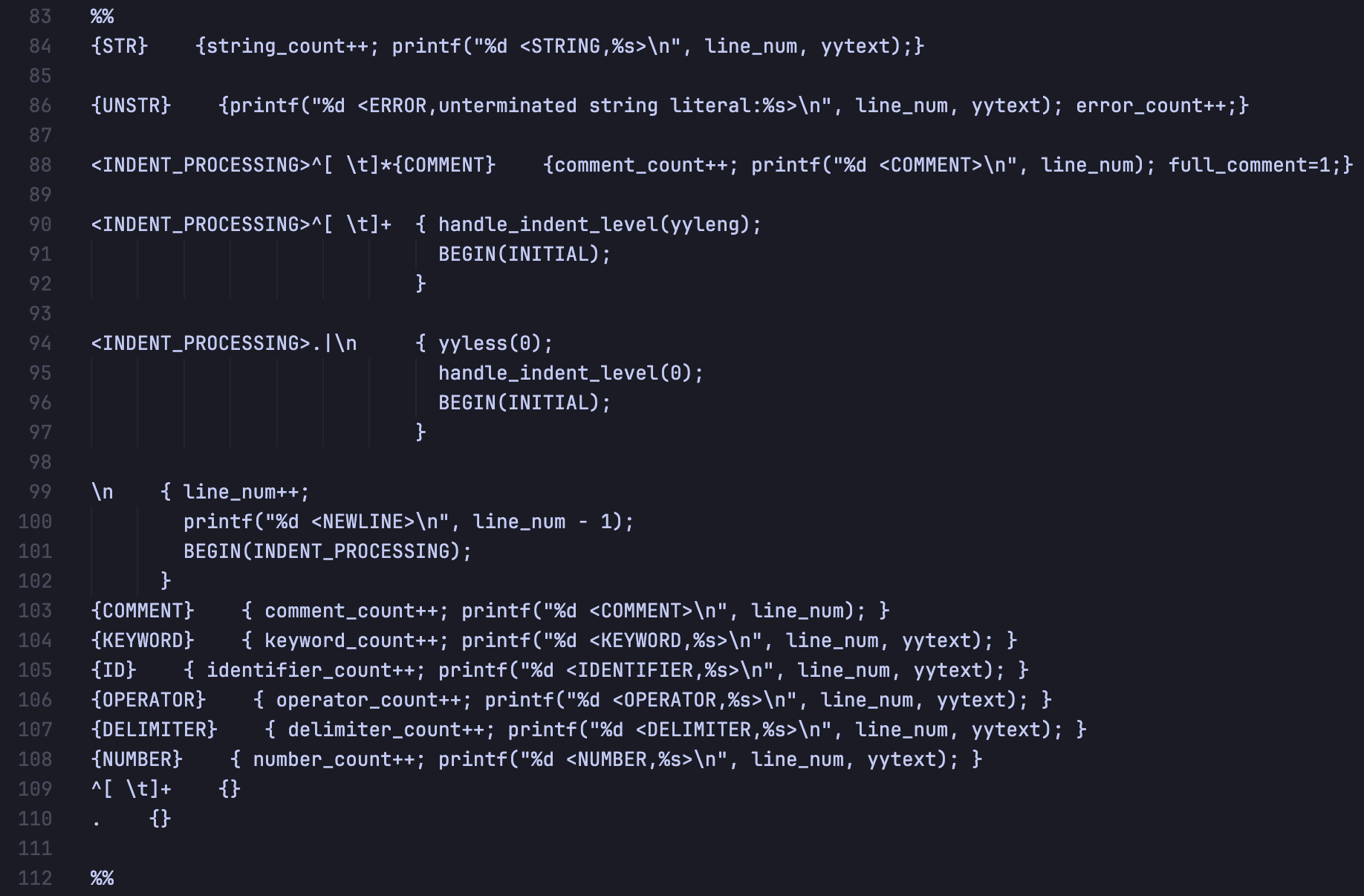
图 2‑4 字符串产生式

1. 数值常量产生式中，指数部分optional\_exponent所能推出的成分中未包含小写e。

图 2‑5 数值常量产生式

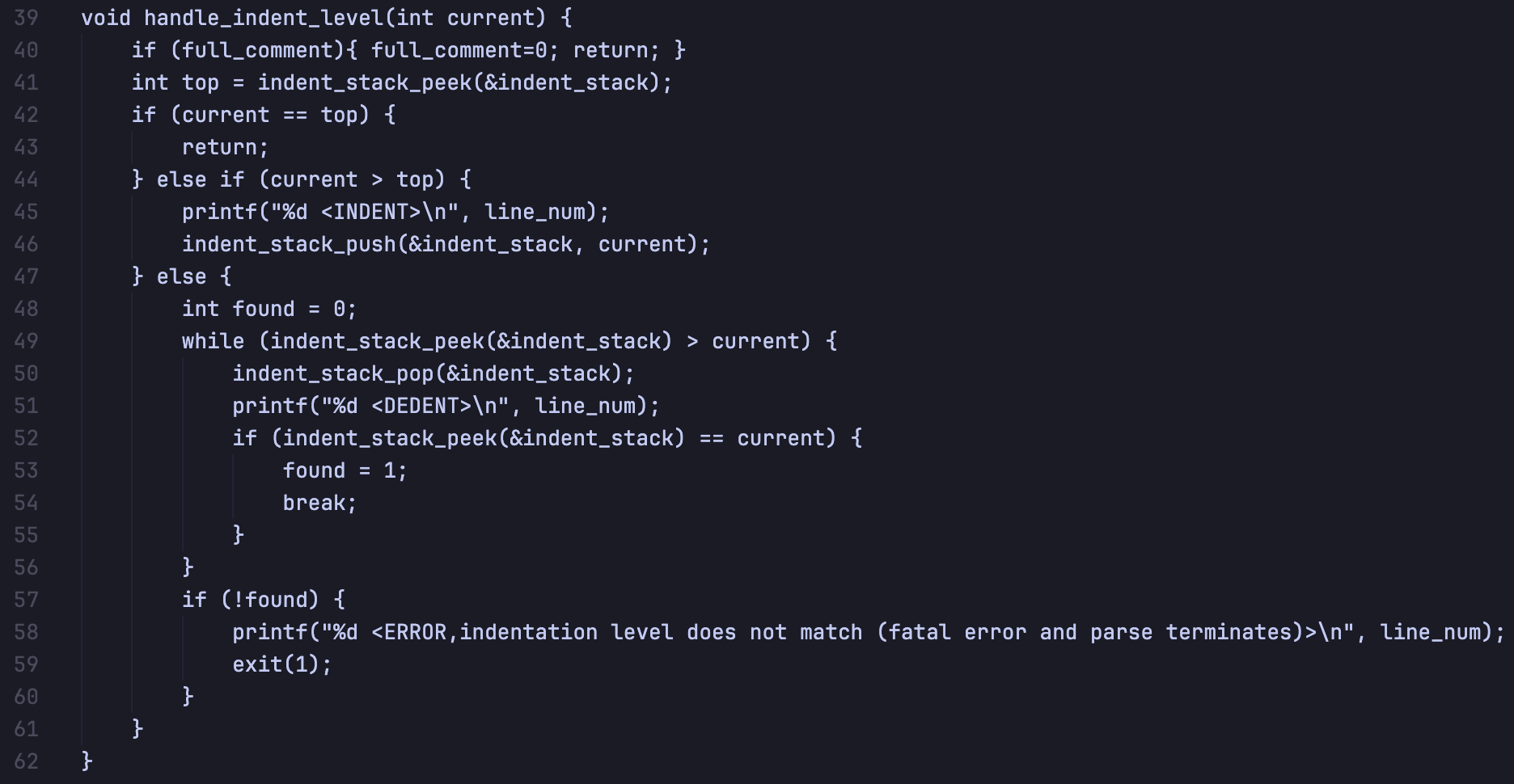
除（在不断提交和debug过程中）修改规则定义，由于产生式中ε无法直接用代码实现，故包含ε的规则如 (a | b |ε) 可用 (a | b)\* 代替；此处将合规字符串与不合规字符串分开定义，便于引号匹配逻辑处理的设计。

## 规则执行

图 2‑6 规则执行代码

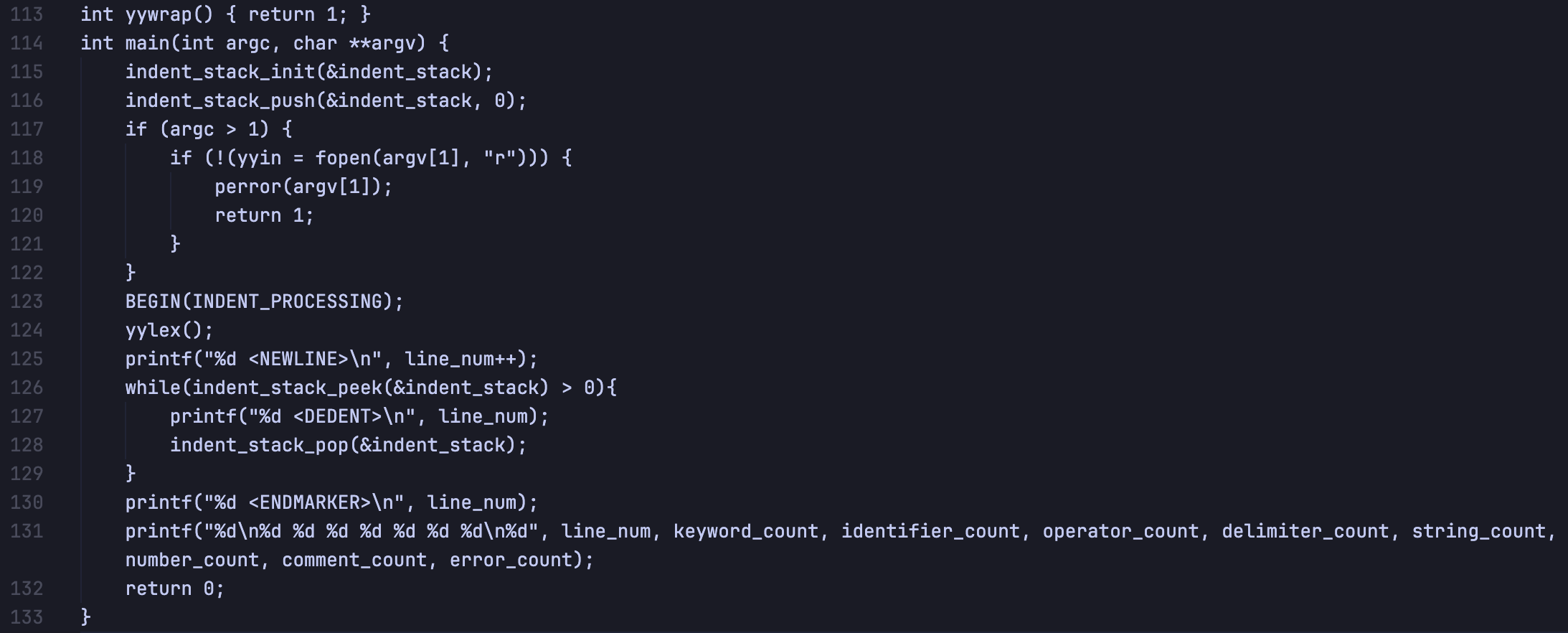
左侧为规则对应的正则表达式形式定义，右侧为正则表达式匹配到后执行的操作，条目声明的先后代表处理优先级。

具体而言，匹配到关键字（KEYWORD）、标识符（IDENTIFIER）、运算符（OPERATOR）、分隔符（DELIMITER）、数值常量（NUMBER）和非全行注释（COMMENT）时，执行计数器+1与输出记录操作；识别到合规字符串（两侧都有双引号）时执行计数器+1与输出记录操作，识别到不合规字符串（只有左侧有双引号）时输出错误信息。在每次识别到换行符后，输出换行记录，进入下一逻辑行，然后立即进行缩进处理操作。

图 2‑7 预定义的缩进处理函数

缩进处理逻辑中，首先识别是否有全行注释，如有，无论注释前缀含有多少空格与缩进，统一忽略。要想实现这一处理逻辑，需要full\_comment这一全局布尔变量记录是否匹配到全行注释，如有则置为真值，分析器输出COMMENT记录后进入 . | \n 处理逻辑中，执行handle\_indent\_level() 函数，此时full\_comment为真，进入函数后不执行任何输出操作，full\_comment重新置为假。如果不含全行注释，则对空格和缩进进行匹配，通过判断缩进长度与缩进栈顶大小关系决定是入栈（输出INDENT记录）还是出栈（输出DEDENT记录）。如果出栈时栈中没有与当前缩进长度相同元素，则输出indentation error。

## 主函数

图 2‑8 主函数

包括文件流读取、缩进栈初始化（向栈底推入0代表无缩进）、第一行缩进处理、词法分析逻辑、行尾换行与文件终止处理逻辑以及输出统计。

# C++实现

C++实现从更底层进行词法分析逻辑的编写，需要考虑更多判定顺序与特殊情况。

## 全局定义

图 3‑1 全局定义

引入完成操作所必要的库、计数器、逻辑行数指示变量、特判用布尔变量、缩进栈结构体、非正则表达式成分的规则（用集合表示）以及缩进处理函数。

## 主函数

图 3‑2 主函数

主函数包括文件流读取、缩进栈初始化、行处理逻辑、文件末处理逻辑以及输出统计。

行处理逻辑包括两部分。一是每次读取新行时的缩进判断，如果为全注释行则无需判断此行，如果不是则进行正常缩进判断和processLine() 行处理函数中集成的逻辑判断。

## 行处理函数processLine()

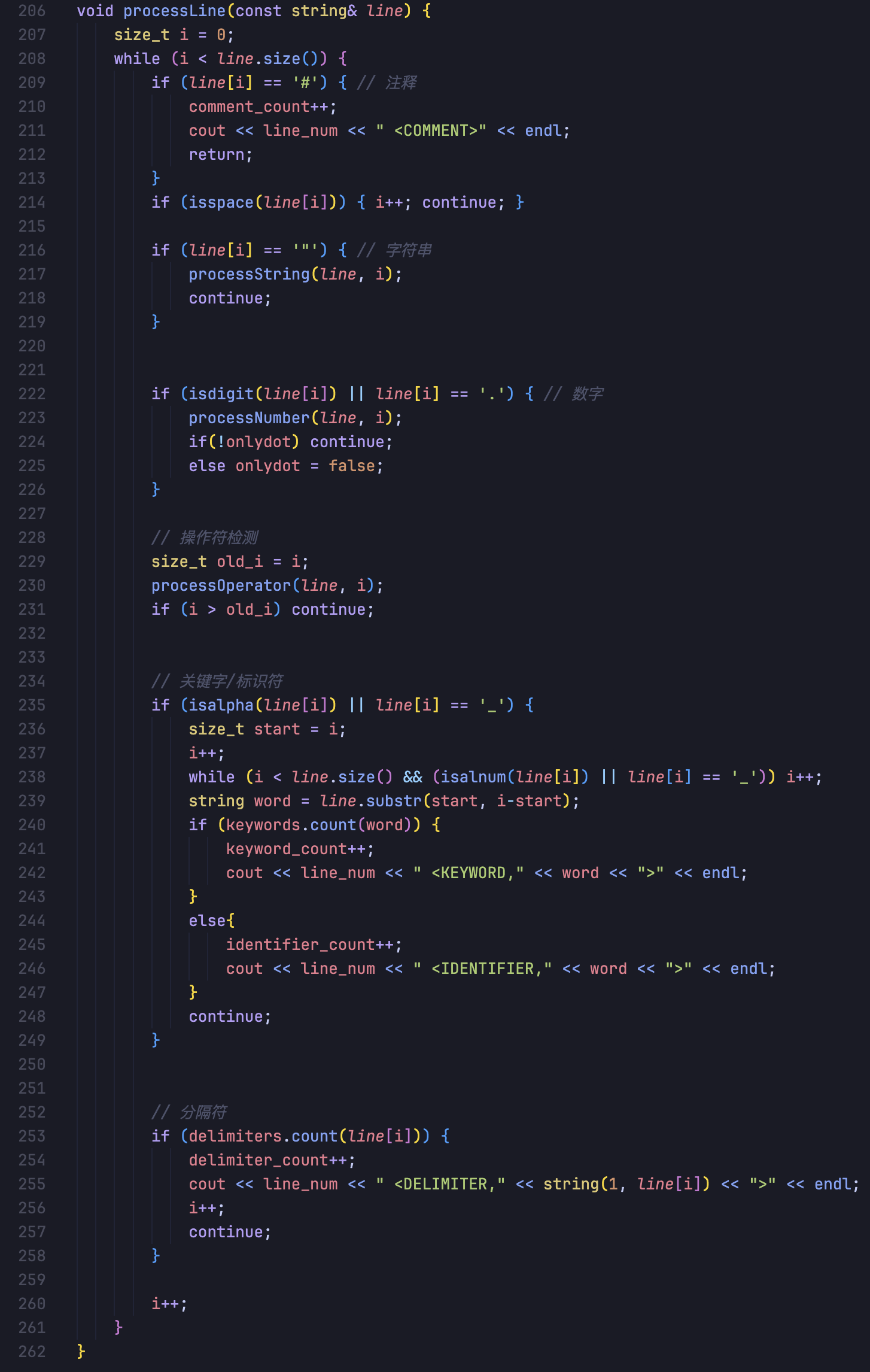
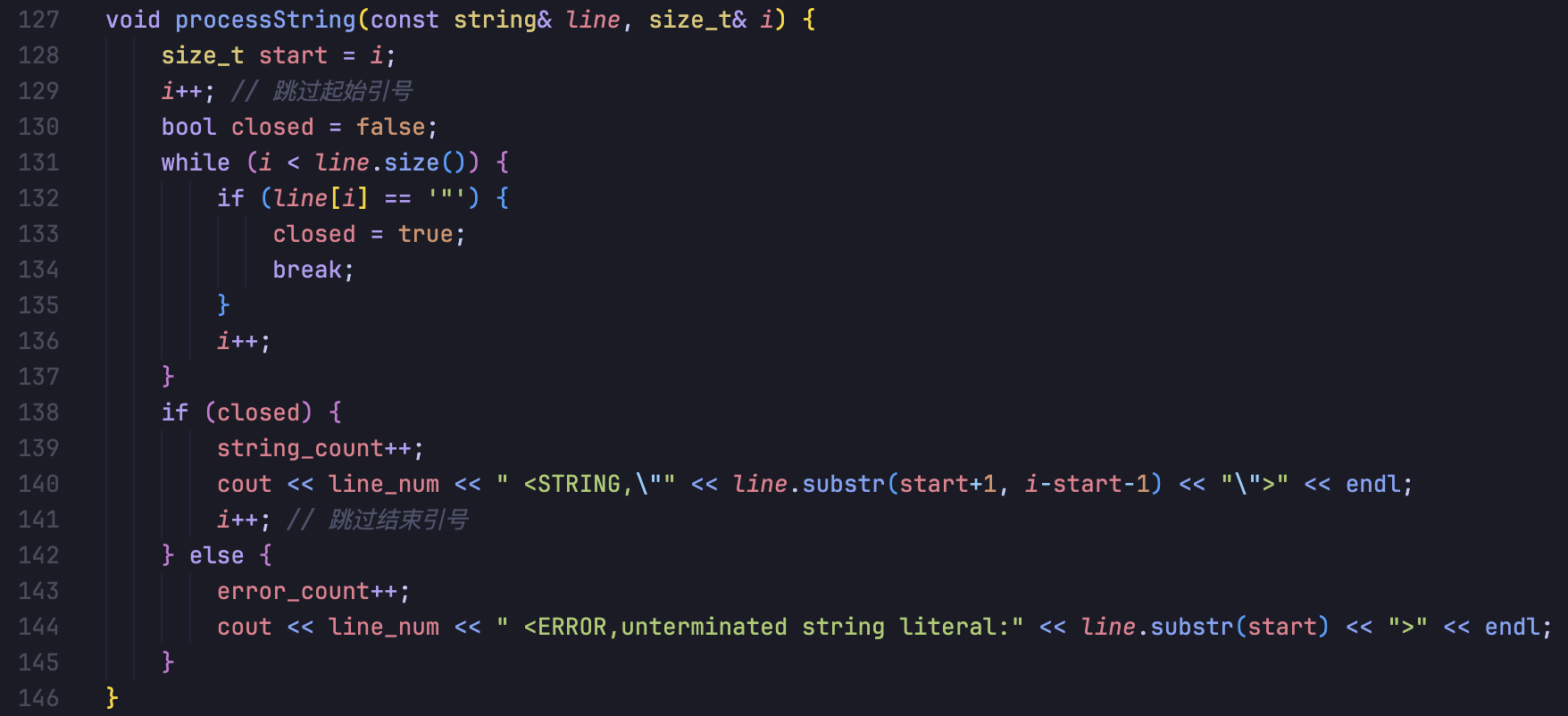


图 3‑3 行处理函数

行处理函数包含一行代码在顺序读取时的处理逻辑。如果识别到 # 号则表明后面均为注释，此时可直接输出COMMENT记录并进入下一行。如果识别到行中空格，无需按缩进处理。如果识别到双引号，则进入字符串处理逻辑；如果识别到数字或 . 号，则进入数值常量处理逻辑；操作符检测逻辑较为特殊，在下一节进行详细说明；如果识别到字母或下划线，进入关键字与标识符检测逻辑；如果硬匹配到分隔符，则输出分隔符记录。

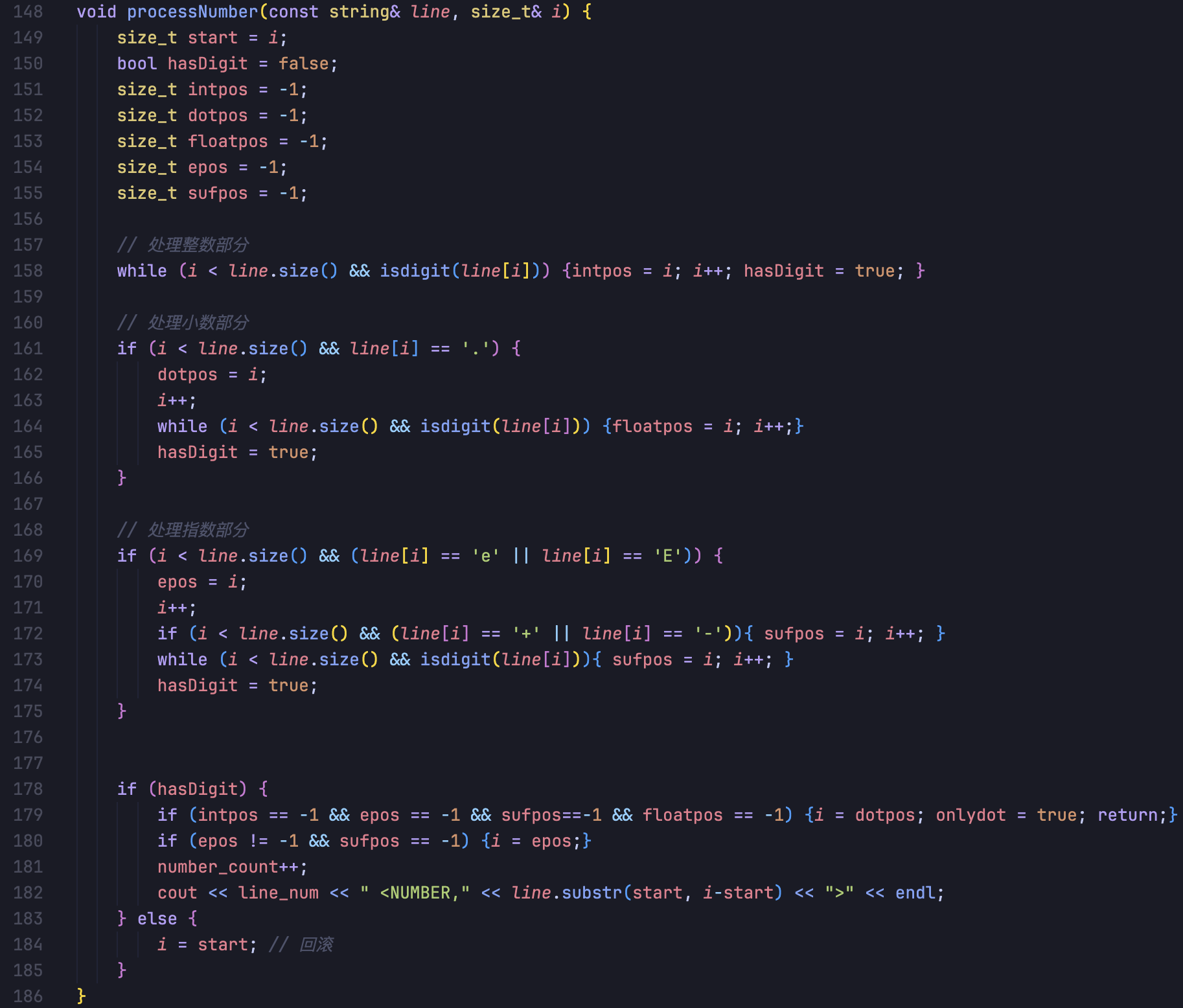
## 各类处理逻辑

### 字符串处理逻辑 processString()

图 3‑4 字符串处理函数

从识别到引号开始，向后搜索是否有匹配引号，如有则输出记录，如没有则输出错误信息。

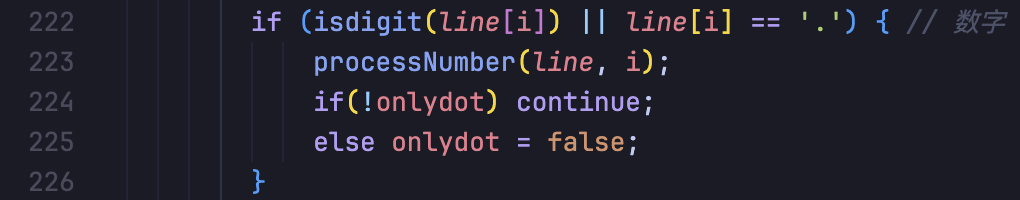
### 数值常量处理逻辑 processNumber()

图 3‑5 字符串处理函数

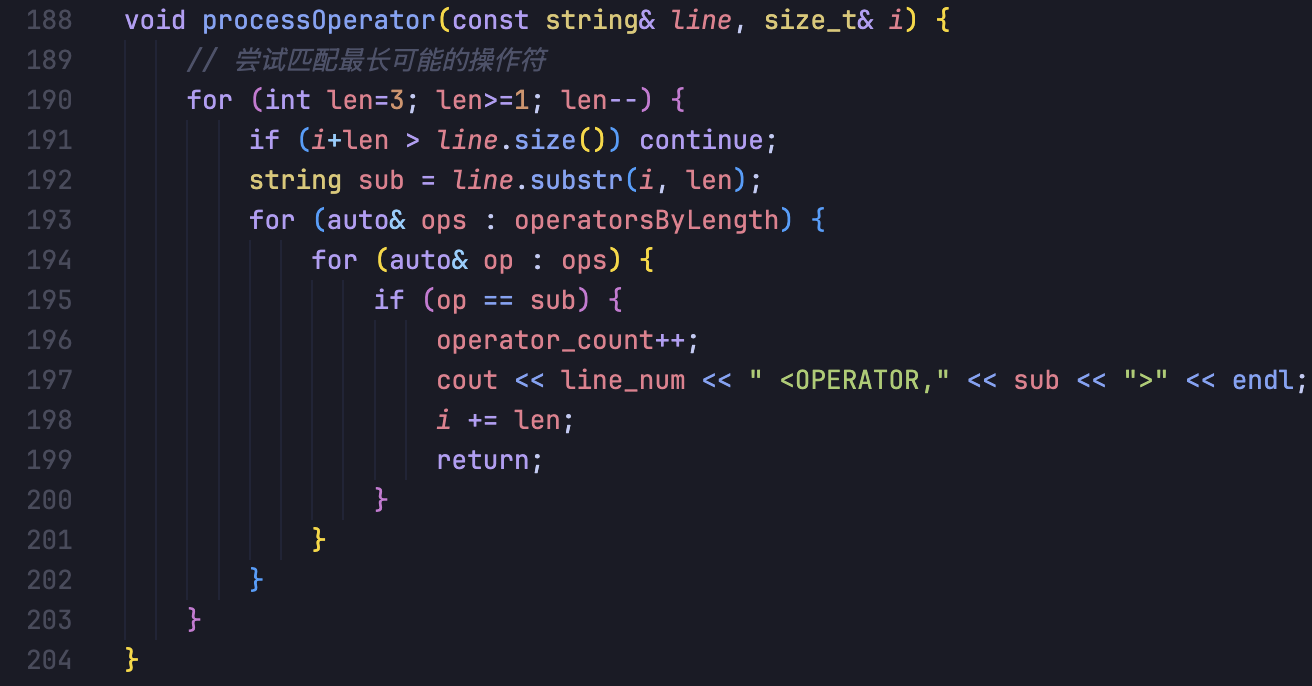
数值常量处理函数中除识别位置之外，还有多个局部变量。hasDigit用来指示是否识别到数字，intpos用于指示整数部分开始的位置，dotpos用于指示小数点位置，floatpos用于指示浮点数部分开始位置，epos用于指示指数标志位置，sufpos用于指示指数后缀起始位置。

首先处理整数部分，从开头对数字顺序读取，然后识别小数点以及后续小数部分，然后处理指数部分。处理完毕后，需要注意即使hasDigit为真，但不一定为合规数值常量，特例有 .（只有小数点），4e（指数无后缀），需要进行特判。

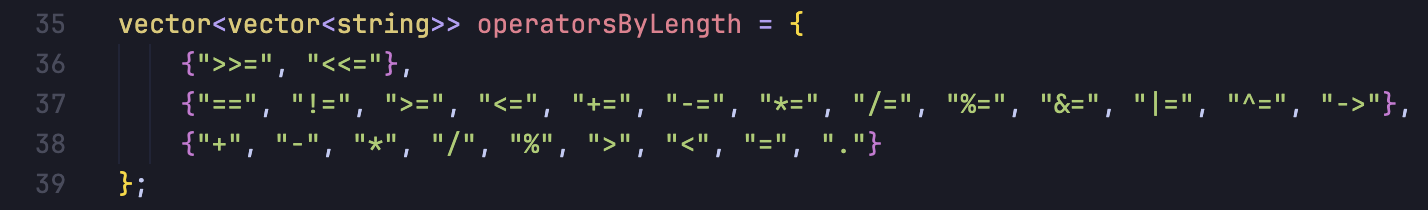
特判过程中，通过各部分位置指示是否为 -1来对特定形式进行筛选。指数无后缀，则有epos不为 -1但sufpos为 -1，此时将全局位置指针移至epos，从e开始进行后续判断。纯小数点特点为intpos、epos、sufpos、floatpos均为 -1，此时将全局位置指针移至dotpos，特殊全局变量onlydot设为真，退出函数后，如果onlydot真则不进入下一次循环，而是继续进入后续判断。如果进入下一次循环，则还会从小数点开始判断，从而进入死循环。

图 3‑6 行处理函数中的字符串处理部分

### 运算符处理逻辑 processOperator()

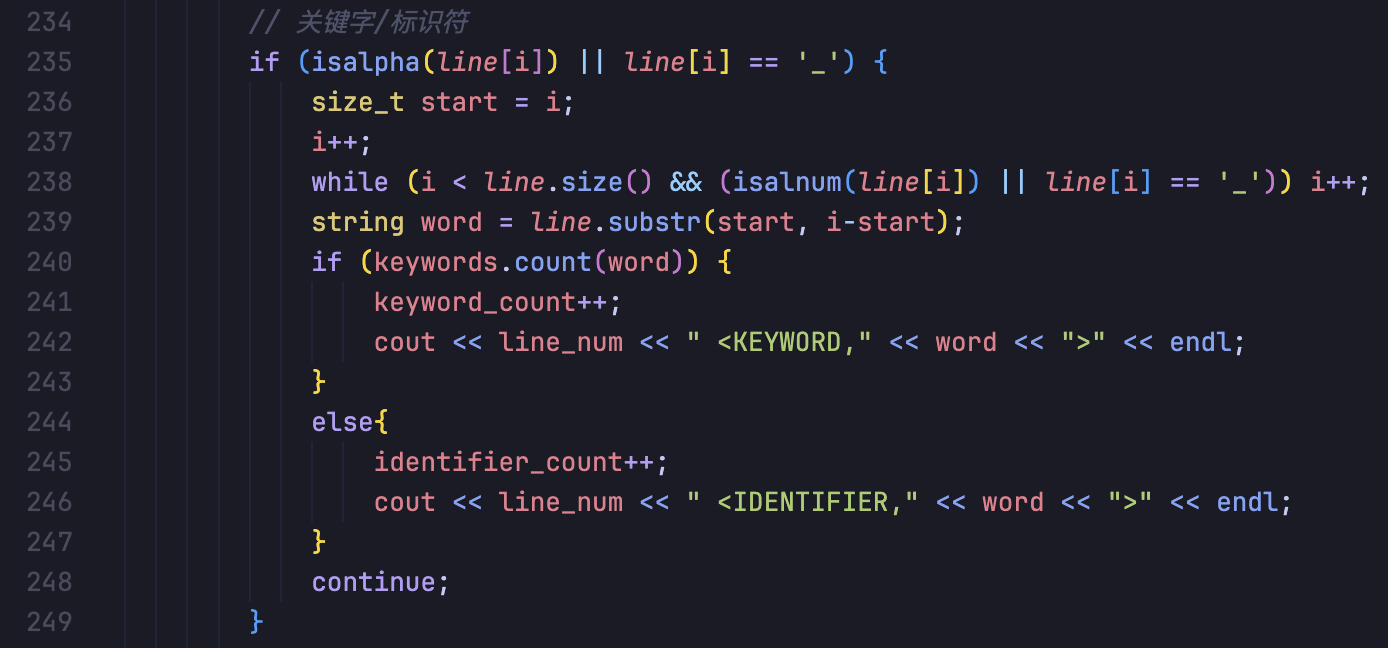
图 3‑7 运算符处理函数

运算符处理部分，由于部分字符需要超前扫描，所以在运算符定义部分按长度对运算符进行了归类：

图 3‑8 按长度区分的运算符定义

运算符最长为3，在匹配时从当前位置向后3个字符的子串开始匹配，不断缩减子串长度，匹配到则输出记录。

### 关键字与标识符处理逻辑

图 3‑9 关键字与标识符处理逻辑

标识符的正则表达式包含关键字，因此关键字匹配要先于标识符。

## 特例分析

1. 在debug过程中，发现and、or、not三者按照标识符处理，而非运算符，与平台实验说明冲突

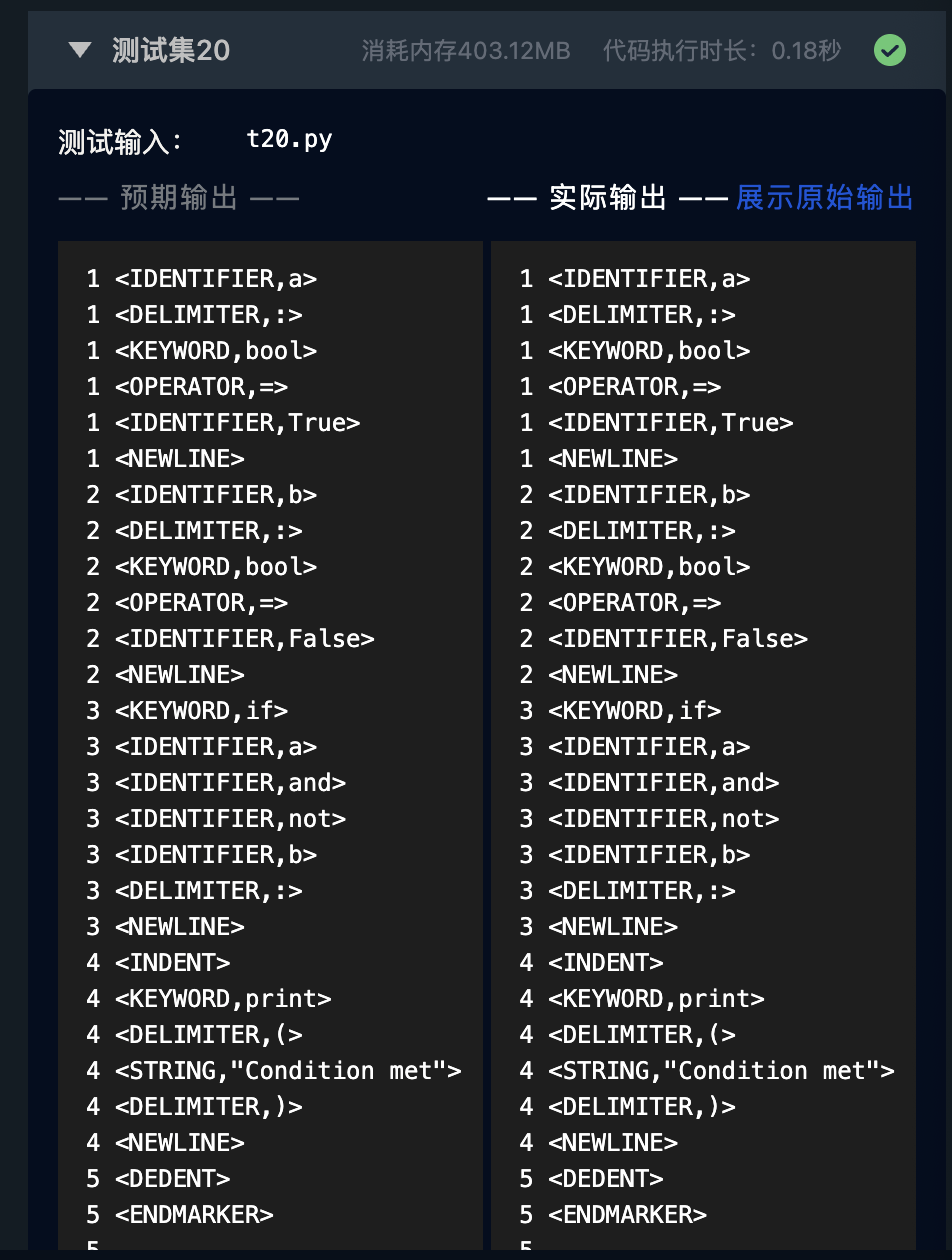


图 3‑10 第三行and与not按标识符输出

1. 测试输入12展示了数值常量判断的各种情况，其中除4.a、3e外均为合法数值常量



图 3‑10 测试用例12及输出

# 线上测试用例通过情况

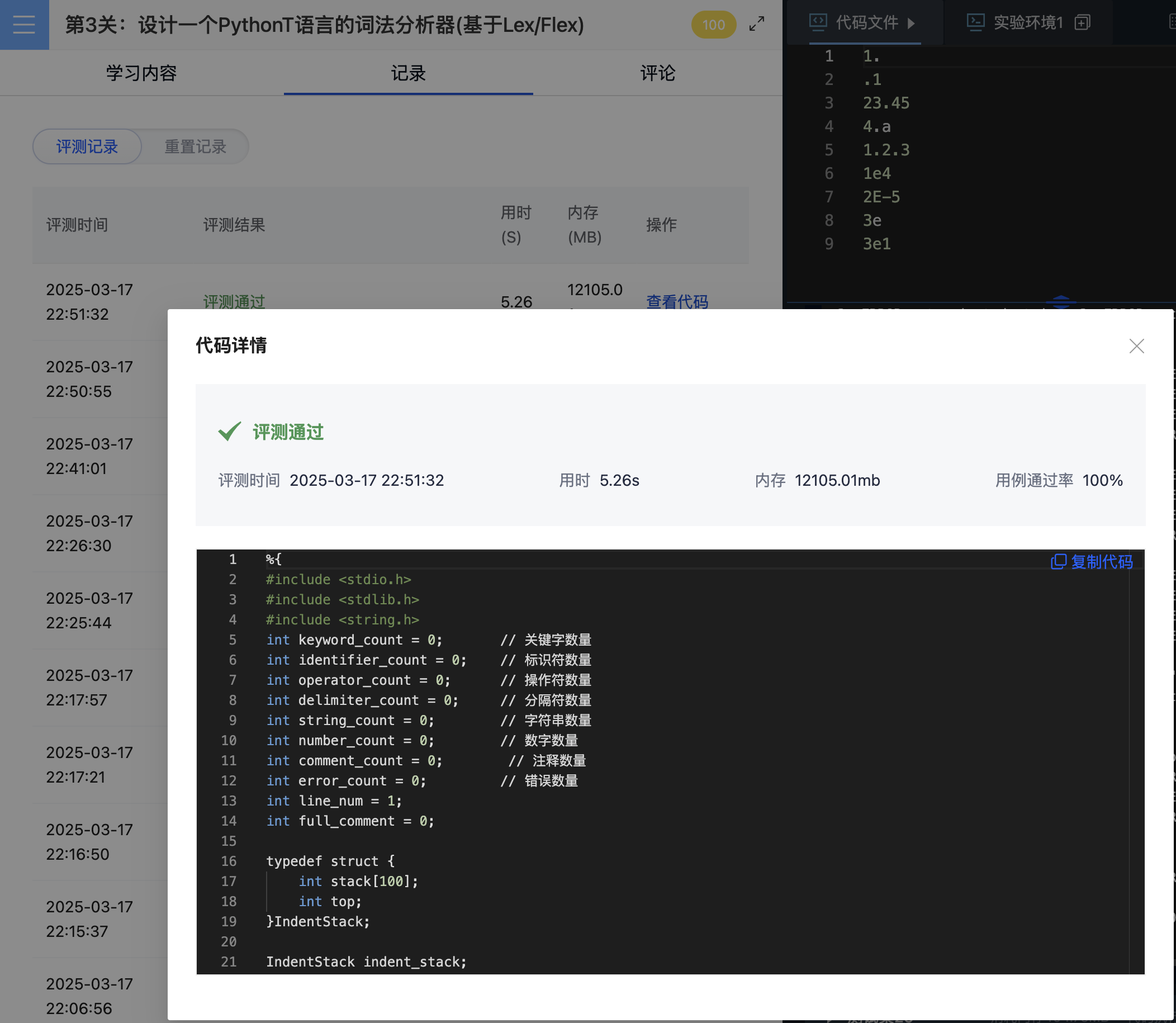


图 4‑1 lex程序通过情况

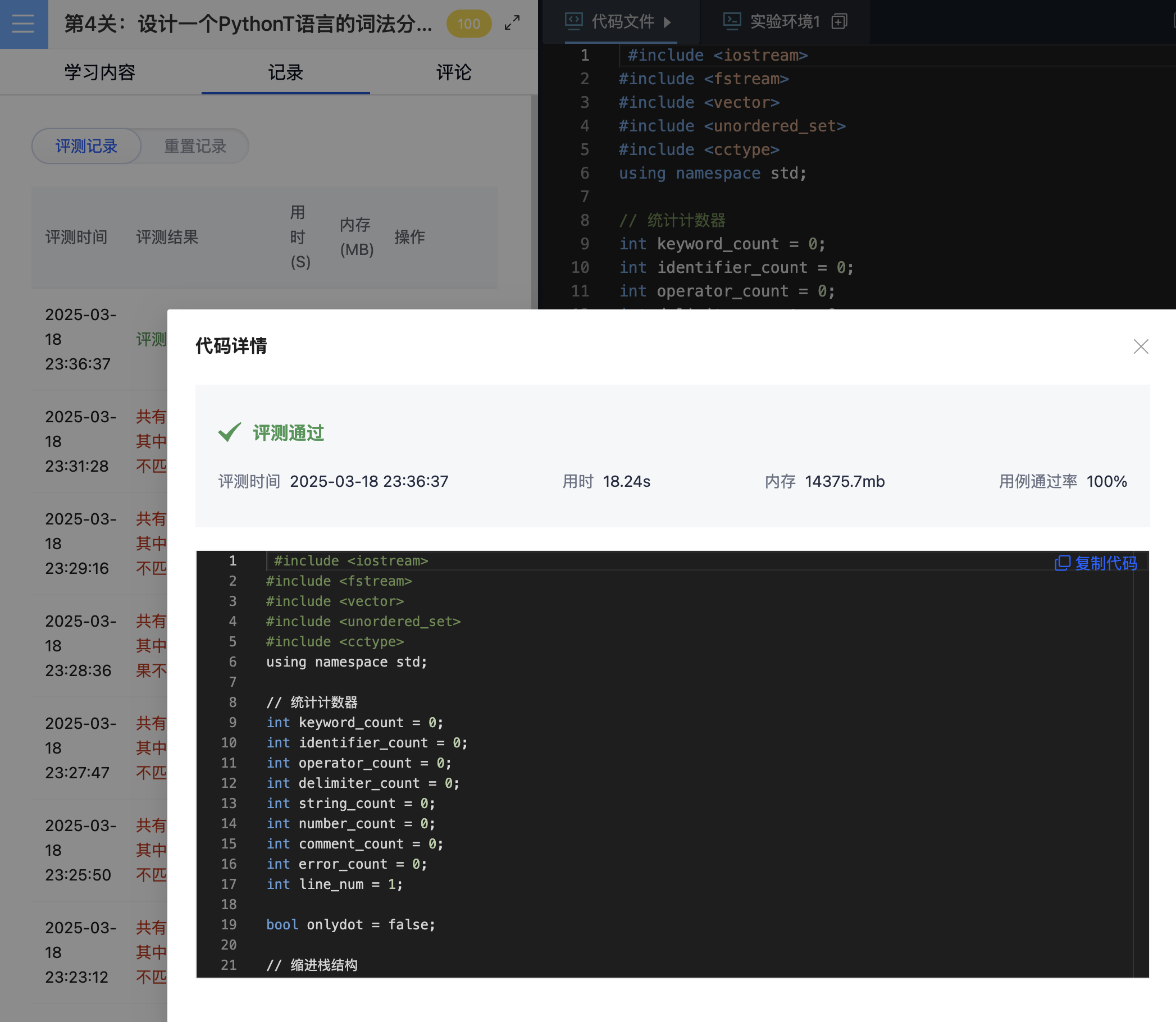


图 4‑2 C++程序通过情况

# 实验总结

本次实验用时接近十小时，期间是不断试错并修正的过程。在这个过程中，我发现了自己在词法分析过程中有时未能考虑较为极端的情况，对lex的熟练程度也有待加强。通过本次实验，我对词法分析有了较好的掌握，进一步深化了对表达式匹配、分析优先级的理解，也深刻意识到了编译原理实验过程中全面思考的重要性。