苏 州 大 学 实 验 报 告

院系	计算机学院	年级专业	21 计科	姓名	方浩楠	学号	2127405048
课程名称	编译原理课程实践					成绩	
指导教师	王中卿	同组实验者	无		实验日期		

实 验 名 称 基于 PLY 的四则运算词法单元识别

一. 实验目的

理解词法分析的基本概念和过程。
学习使用 PLY（Python Lex-Yacc）库构建词法分析器。
能够识别并标记化 C++ 语言的基本子集。

二. 实验内容

读取 prog.txt 中的内容,使用 ply,来识别 prog.txt 中所有的词法单元

三. 实验步骤和结果
实验代码在 **source/main.py** 中

环境准备:
安装 Python 3.x。
安装 PLY 库: 在命令行中执行 pip install ply。
创建词法分析器:
新建一个名为 lexer.py 的文件。
在 lexer.py 中，按照前面提供的代码示例创建词法分析器，定义识别的符号及其对应的正则表达式规则。
准备测试数据:
创建一个名为 prog.txt 的文本文件。
在 prog.txt 中输入一些 C++ 代码，作为测试数据。
运行词法分析器:
在命令行中导航到 lexer.py 文件的位置。
执行命令 python lexer.py，运行词法分析器。
观察并记录输出的标记。
结果分析:
分析输出的标记，验证它们是否与 prog.txt 中的代码相匹配。
检查是否所有的符号都被正确识别并标记化。
错误处理和调试:
如有必要，调整词法分析器的代码以修复任何错误或问题。
重复步骤 4 和 5，直到得到满意的结果。

扩展实验:
（可选）尝试添加更多符号和规则，以识别更多的 C++ 语法特性。
重复步骤 4 和 5，测试新的符号和规则。

四. 实验总结

通过本次实验，我们深入理解了词法分析的基本概念和重要性。词法分析是编译过程的第一步，它将源代码转换为一系列的标记，为语法分析阶段做了准备。以下是本次实验的主要收获和发现：
理论与实践相结合：
通过实际编写词法分析器代码，我们将理论知识应用于实践，加深了对词法分析过程的理解。
掌握 PLY 工具的使用：
PLY 是一个强大的 Python Lex-Yacc 库，通过本次实验，我们学会了如何使用 PLY 定义词法规则和构建词法分析器。
正则表达式的应用：
正则表达式是定义词法规则的关键。通过实验，我们练习了如何编写正则表达式来匹配不同的词法符号。
错误处理和调试：
在实验过程中，我们遇到了一些错误和问题，例如最初忘记为 << 运算符定义规则。通过调试和修复这些问题，我们学会了如何改进词法分析器，并确保它能正确地识别所有期望的符号。
实验拓展与探索：
我们尝试添加更多的词法规则，以处理 C++ 语法的更多特性，这增强了我们对词法分析复杂性的理解，并为未来的学习和探索提供了基础。

```
import ply.lex as lex

reserved = {
    'while': 'WHILE',
    'if': 'IF',
    'cout': 'COUT',
}


# List of token names
tokens = (
    'INT',
    'ID',
    'EQUALS',
    'NUMBER',
    'PLUS',
    'MINUS',
    'MULTIPLY',
    'DIVIDE',
    'LPAREN',
    'RPAREN',
    'LBRACE',
```

```
    'RBRACE',
    'LT',
    'STRING',
    'SEMICOLON',
    'COMMA',
    'ENDL',
    'LSHIFT',
    *reserved.values(),
)

# Regular expression rules for simple tokens
t_EQUALS = r'='
t_PLUS = r'\+'
t_MINUS = r'-'
t_MULTIPLY = r'\*'
t_DIVIDE = r'/'
t_LPAREN = r'\('
t_RPAREN = r'\)'
t_LBRACE = r'\{'
t_RBRACE = r'\}'
t_LT = r'<'
t_SEMICOLON = r';'
t_COMMA = r','
t_ENDL = r'endl'
t_LSHIFT = r'<<'


# A regular expression rule with some action code
def t_NUMBER(t):
    r'\d+'
    t.value = int(t.value)
    return t


def t_STRING(t):
    r'\".*?\"'
    return t


def t_ID(t):
    r'[a-zA-Z_][a-zA-Z_0-9]*'
    t.type = reserved.get(t.value, 'ID')
    return t
```

```
def t_newline(t):
    r'\n+'
    t.lexer.lineno += len(t.value)


t_ignore = ' \t'


def t_error(t):
    print("Illegal character '%s'" % t.value[0])
    t.lexer.skip(1)


# Build the lexer
lexer = lex.lex()

# Test it out
with open('prog.txt', 'r') as f:
    data = f.read()

# Give the lexer some input
lexer.input(data)

# Tokenize
while True:
    tok = lexer.token()
    if not tok:
        break
    print(tok)
```


教务处制