《编译原理》

——第一次实验报告

学　号：　15281106

姓　名：　徐开元

专　业：　计算机科学与技术

学　院：　计算机与信息技术学院

提交日期：　2018 年 4 月 7 日

指导老师：　徐金安

一、实验内容:

[实验项目] 以下为正则文法所描述的 C 语言子集单词符号的示例，请补充单词符号: ++, --, >>, <<, += , -= ,*=, /= ， &&(逻辑与)，||(逻辑或)，!(逻辑非)等等，给出补 充后描述 C 语言子集单词符号的正则文法，设计并实现其词法分析程序。

<标识符>→字母— <标识符>字母— <标识符>数字

<无符号整数>→数字— <无符号整数>数字

<单字符分界符> →+ —- —* —;—, —(—) —{—} <双字符分界符>→<大于>=—<小于>=—<小于>>—<感叹号>=—<等于>=—<斜竖>*

<小于>→< <等于>→= <大于>→> <斜竖> →/

<感叹号>→!

该语言的保留字 :void、int、float、double、if、else、for、do、while 等等(也可补充)。

[设计说明] (1)可将该语言设计成大小写不敏感，也可设计成大小写敏感，用户定 义的标识符最长不超过 32 个字符;(2)字母为 a-z A-Z, 数字为 0-9;(3)可以对上述文法 进行扩充和改造;(4) “/*.....*/” 和 “//” (一行内)为程序的注释部分。

[设计要求] (1)给出各单词符号的类别编码;(2)词法分析程序应能发现输入串中的 错误;(3)词法分析作为单独一遍编写, 词法分析结果为二元式序列组成的中间文件;(4) 设计两个测试用例(尽可能完备)，并给出测试结果。

二、程序描述

2.1 扩充并改造后的正则文法:

<标识符>→c|c<余留标识符>

<余留标识符>→d|c

<无符号数>→d<余留无符号数>|.<小数部分>|d

<余留无符号数>→d<余留无符号数>|.<十进小数>|(E|e)<指数部分>|.|d

<十进小数>→(E|e)<指数部分>|d<十进小数>|d

<小数部分>→d<十进小数>|d

<指数部分>→d<余留指数>|(+|-)<整指数>|d

<整指数>→d<余留整指数>|d

<余留整指数>→d<余留整指数>|d

<算数运算符>→+|-|*|/|++|--

<关系运算符>→>|<|==|>=|<=|!=

<逻辑运算符>→!|&&|\|\|

<位操作运算符>→>>|<<

<赋值运算符>→=|+=|-=|*=|/=|%=

<特殊运算符>→,|\(|\)|{|}

<分隔符>→;

保留字从网站中获取，并包含了大部分 c 语言程序保留字。

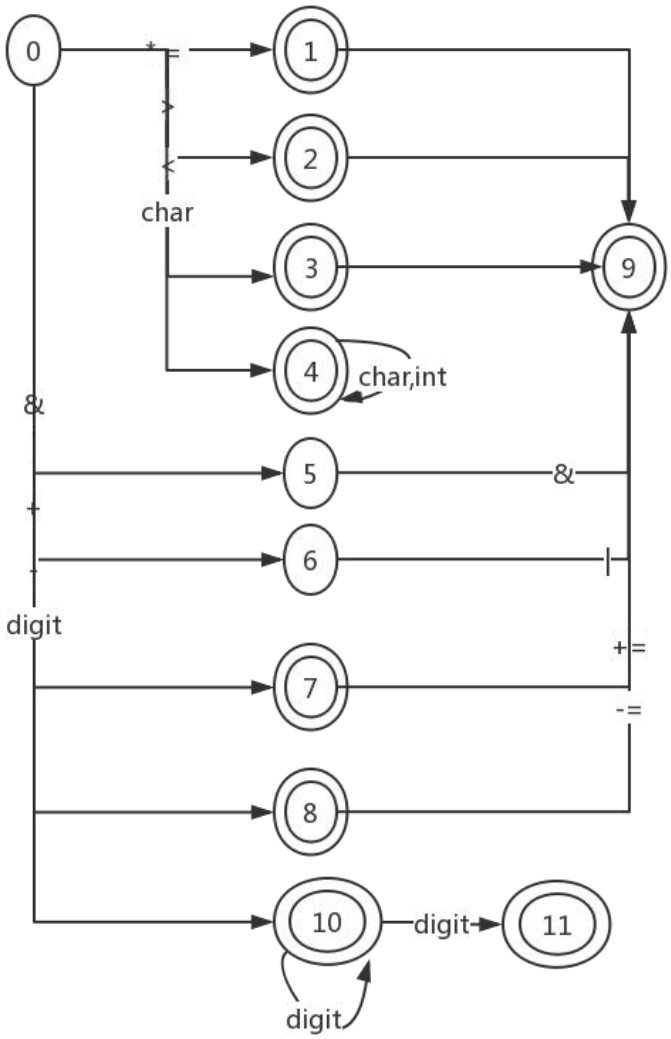
2.2 程序识别的关键字及识别码

```
Int = 1, "int"
Long = 2, "long"
Short = 3, "short"
Float = 4, "float"
Double = 5, "double"
Char = 6, "char"
Unsigned = 7, "unsigned"
Signed = 8, "signed"
Const = 9, "const"
Void = 10, "void"
Volatile = 11, "volatile"
Enum = 12, "enum"
Struct = 13, "struct"
Union = 14, "union"
If = 15, "if"
Else = 16, "else"
Goto = 17, "goto"
Switch = 18, "switch"
Case = 19, "case"
Do = 20, "do"
While = 21, "while"
For = 22, "for"
Continue = 23, "continue"
```

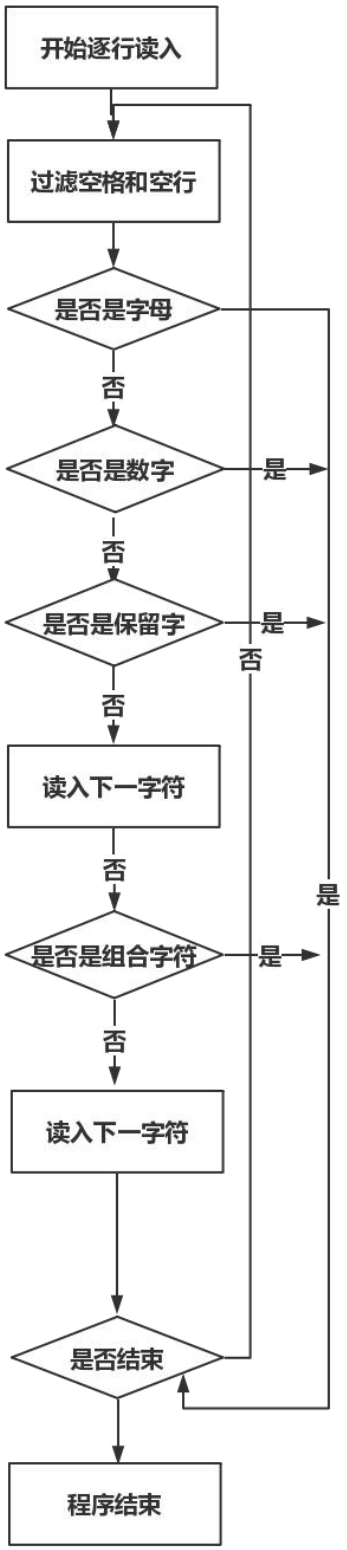
```
Break = 24, "break"
Return = 25, "return"
Default = 26, "default"
Typeof = 27, "typeof"
Auto = 28, "auto"
Register = 29, "register"
Extern = 30, "extern"
Static = 31, "static"
Sizeof = 32, "sizeof"
OpenBlock = 33, "{"
CloseBlock = 34, "}"
OpenParen = 35, "("
CloseParen = 36, ")"
OpenBracket = 37, "["
CloseBracket = 38, "]"
Dot = 39, "."
Semicolon = 40, ";"
Comma = 41, ","
LessThan = 42, "<"
GreaterThan = 43, ">"
LessThanOrEq = 44, "<="
GreaterThanOrEq = 45, ">="
Eq = 46, "=="
NotEq = 47, "!="
Add = 48, "+"
Sub = 49, "-"
Mul = 50, "*"
Div = 51, "/"
Mod = 52, "%"
Inc = 53, "++"
Dec = 54, "--"
LeftSh = 55, "<<"
RightSh = 56, ">>"
And = 57, "&"
Or = 58, "|"
Xor = 59, "^"
Not = 60, "!"
Neg = 61, "~"
BoolAnd = 62, "&&"
BoolOr = 63, "||"
```

```
Question = 64, "?"
Colon = 65, ":"
Assign = 66, "="
AssignAdd = 67, "+="
AssignSub = 68, "-="
AssignMul = 69, "*="
AssignDiv = 70, "/="
AssignMod = 71, "%="
AssignLeftSh = 72, "<<="
AssignRightSh = 73, ">>="
AssignAnd = 74, "&="
AssignOr = 75, "|="
AssignXor = 76, "^="
Arrow = 77, "->"
EOF = 78,
Identifier = 79,
NullLiteral = 80, "null"
NumericLiteral = 81,
StringLiteral = 82,
Comment = 83
```

2.3 状态转换图： （DFA M）



2.4 程序运行流程图


开始逐行读入
过滤空格和空行
是否是字母
否
是否是数字
是
否
是否是保留字
是
否
否
读入下一字符
否
是否是组合字符
是
否
读入下一字符
是
是否结束
程序结束

2.5 程序示例

在本次实验中，我在 macox 下使用 python3.6.5 实现了词法分析程序（lexer）其中代码目录如下：

keyword 为关键保留字符号表，建立了符号字典，用于分词给出 token。

Lexer 文件为词法分析主类，用于逐行读取分析扫描输入文件每行。

punc 文件为符号字典，用于分词给出 token。

token 文件定义了抽象的 token 大类。

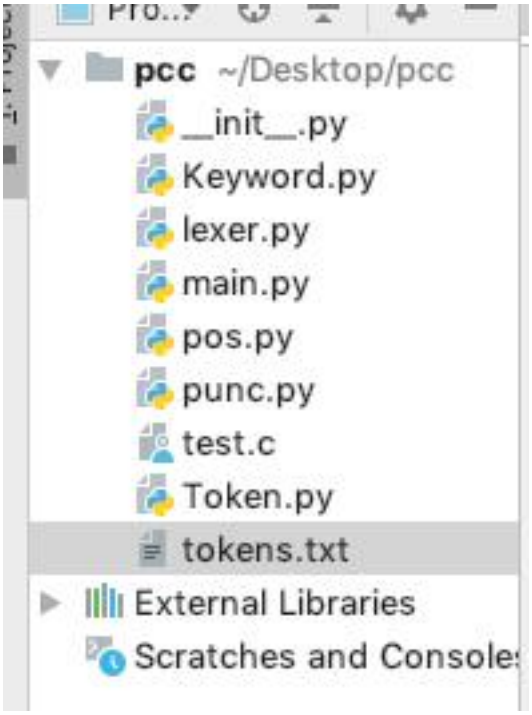
pos 文件定义了词的行列位置。

main 文件为主函数入口。

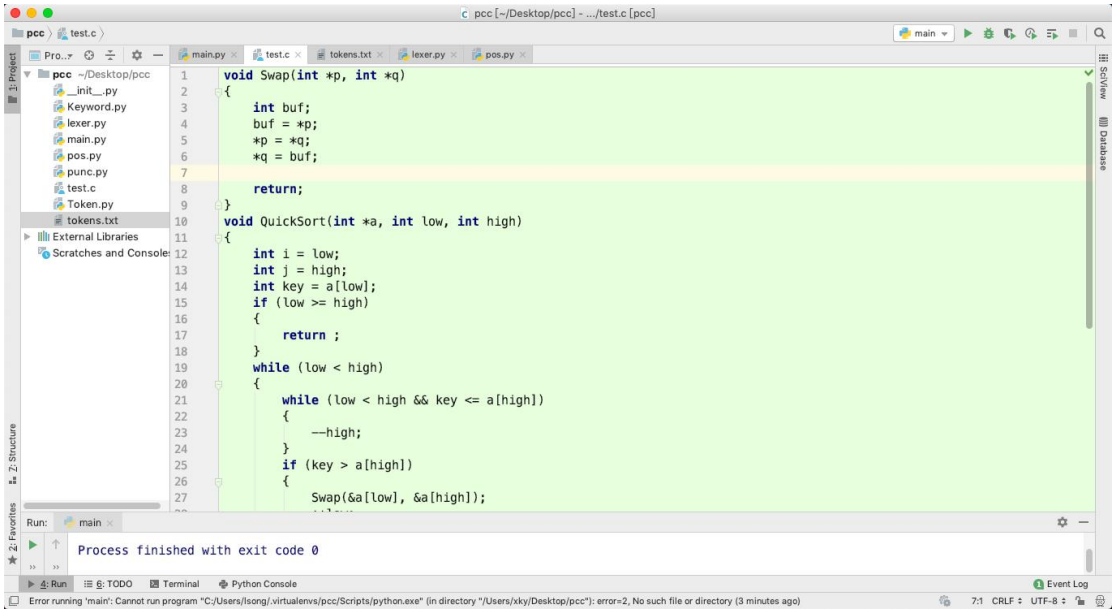
Test.c 为测试 c 语言输入文件。

Token.txt 为输出的 token 文件。

__init__为 python 库初始化文件。

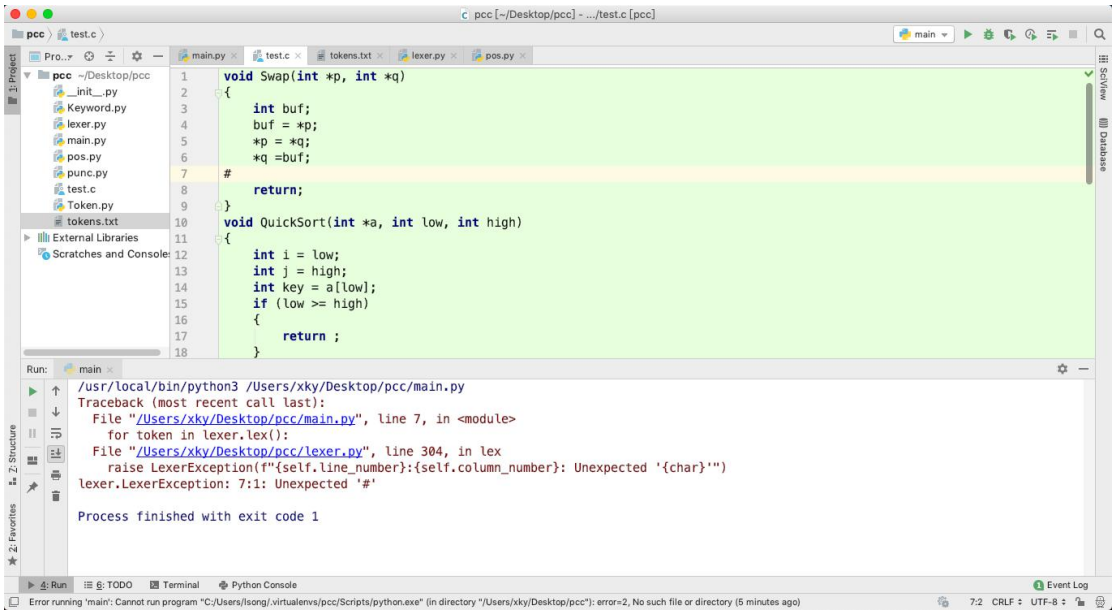


选择测试的样例为 c 语言实现快速排序程序。

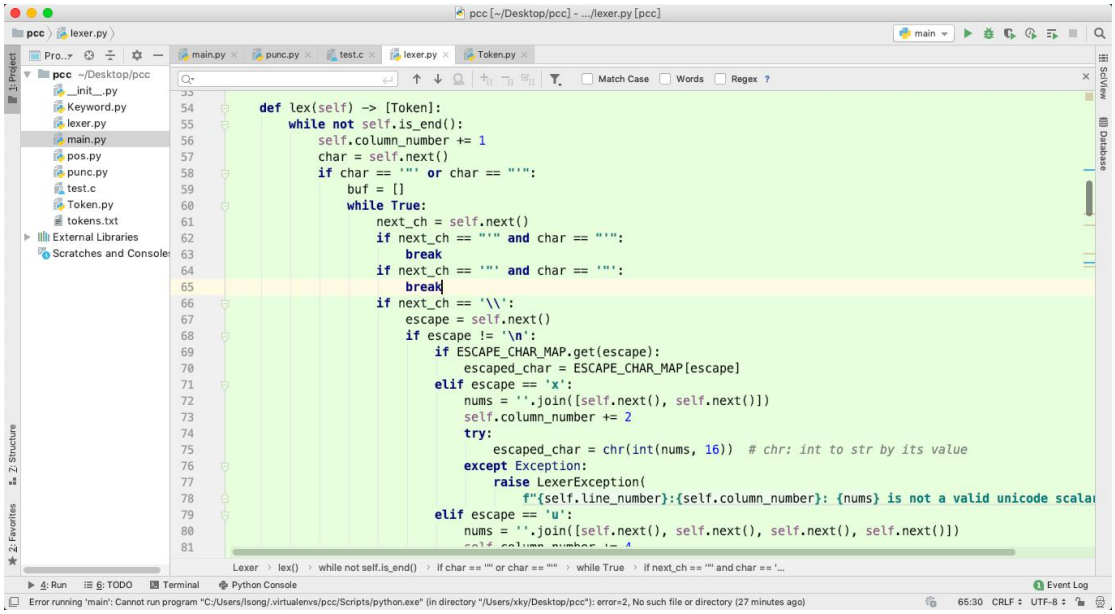


样例文件 test.c

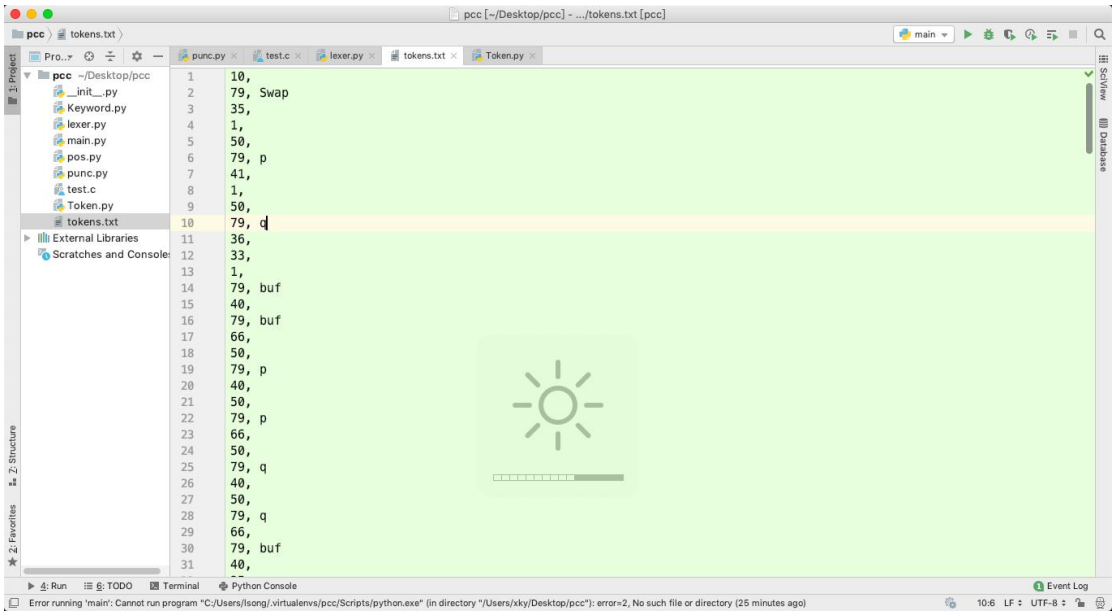
修改样例文件 test.c，检测到错误，并报出错误的行数，错误的字符，抛出自定义错误 LexerException。



修改样例文件 test.c，检测出具体错误为#行



lexer 主类，逐行逐字读取输入文本并判断词类型。



程序输入结果存入 token.txt

三、实验心得

通过此次实验让我了解了如何设计、编制并调试词法分析程序，并加深了我对词法分析器原理的理解，使我更深层次的掌握了词法分析。从刚开始的无从下手到后来渐渐的突破了各个难关，将状态转换图多次修改后我决定采用最简的形式，生成合法字符串后再进行类别划分，这样可以有效减少状态个数，提高效率。虽然期间花了大量的时间和精力，但是让我获益匪浅。以前只是知道高级语言程序的编写，并不清楚底层编译器的具体实现，也一直对编译器这部分内容非常感兴趣。通过这次实验，使用了自己相对熟悉的 python 语言实现了一个相对简单的扫描器，熟悉了直接构造词法分析器的方法和相关原理。对于自己理解程序代码运行的原理和对某一程序的实现优化有了一些想法。