東南大學
SOUTHEAST UNIVERSITY

编译原理实验

报告一

实验名称： Lexical Analyzer

学生姓名： 白丰硕

学生学号： 71116233

东南大学计算机科学与工程学院、软件学院

School of Computer Science & Engineering College of Software Engineering

Southeast University

二 0 一 八 年 十 二 月

東南大學
SOUTHEAST UNIVERSITY

目录

一、 实验目的

(1) 巩固有限自动机理论与正规文法，正规式三者之间的关系和相关的知识点

(2) 了解词法分析程序具体的实现方法及所用算法的原理

(3) 编写相关的词法分析程序实现对 C++语言程序文件的词法分析

二、 实验环境

(1) 开发环境：

- OS：Ubuntu 16.04
- 语言：Python 3.6
- 画图：ProcessOn
- 打包：Pyinstaller

(2) 运行环境

- Windows 或 Linux

三、 实验内容

编写代码实现一个词法分析程序，可以对 C++语言的.cpp 和.h 文件中的单词，如：关键字、操作运算符、界符、标识符、字符、字符串、整数、浮点数、注释等进行识别并提取出相关的单词，形成 token 序列，并对一些可能出现的错误进行相应的处理。

主要功能

➢ 读写文件

词法分析程序可以从指定的文件路径下，读取指定的文件，进行词法分析过程，并将做好的词法分析的 token 流写到词法分析程序本目录下的 token 文件夹中。

➢ 词法分析

词法分析主程序将读进程序的文件字符流开始进行逐个扫描，进行相应的词法分析。通过扫描，找出相应的单词以及其相应的单词类别号和它所属的内码，形成 token 序列，为输出做准备。

➢ 外部参数

本程序中提供外部参数，使得词法分析程序可以更加灵活的分析、运行程序，可以选择任何路径下的文件作为被分析文件，并将生成的 token 序列文件放置于本目录下的 token 文件夹中

参数	描述
-h	show this help message and exit
-f，--file	path of the file to analyze
-p,--print	wether to print the token sequence

➢ 错误处理

类型	描述
浮点数小数点错误	形如：1.2.3
数字混有字母	形如：123w
字符位数过多	形如：‘123’
字符丢失单引号	形如：‘1
字符串丢失引号	形如：“123
注释部分缺失	形如：/*165
标识符命名错误	形如：=w1

不可识别的单词　　形如：α

单词表

- **basicWordTable**

"include","define","auto","bool","break","case","catch","char","class",

"const","const_cast","continue","default","delete","do","double",

"dynamic_cast","else","enum","explicit","extern","false","float","for",

"friend","goto","if","inline","int","long","mutable","namespace","new",

"operator","private","protected","public","register","reinterpret_cast",

"return","short","signed","sizeof","static","static_cast","struct",

"switch","template","this","throw","true","try","typedef","typeid",

"typename","union","unsigned","using","virtual","void","volatile","while"

- **separatorTable**

",",",",";",".","(",")","[","]","{","}","#"

- **operatorTable**

"+", "-","++", "--", "*", "/", "<", "<=", ">", ">=", "=", "==","!="

四、 实验结果

该实验结果由以下这个简短的 C++程序所构造得出，只是为了展示词法分析效果，更多的示例在 sample 文件下可以找到。

```
include<iostream>
float inc(float q){
    return q++;
}
/*
这是一个没什么用的注释
*/
int main(){
    //声明一个变量
    int a = inc(1.2);
}
```

```
======line 1======
(include,0,--)
(<,86,--)
(iostream,500,0)
(>,88,--)
======line 2======
(float,22,--)
(inc,500,1)
((,73,--)
(float,22,--)
(q,500,2)
(),74,--)
({,77,--)
======line 3======
(return,39,--)
(q,500,2)
(++,82,--)
(;,71,--)
======line 4======
(},78,--)
======line 5======
(/*这是一个没什么用的注释*/,200,--)
======line 8======
(int,28,--)
(main,500,3)
((,73,--)
(),74,--)
({,77,--)
======line 9======
(//声明一个变量,200,--)
======line 10======
(int,28,--)
(a,500,4)
(=,90,--)
(inc,500,1)
((,73,--)
(1.2,104,0)
(),74,--)
(;,71,--)
======line 11======
(},78,--)
```

五、 数据结构

名称	类型	描述
basicWordTable	List	放置 C++的大部分关键字
separatorTable	List	放置 C++分隔符
operatorTable	List	放置 C++操作运算符
identitierTable	List	放置标识符
charTable	List	放置被分析程序中的字符变量
stringTable	List	放置被分析程序中的字符串变量
integerTable	List	放置被分析程序中的整数变量
floatTable	List	放置被分析程序中的浮点数变量

六、 实验算法

实验采用 Tompson 算法，Tompson 构造法是计算机科学中将正则表达式转化为一个与之等价的非确定有限自动机的算法。

- **构造正则表达式**

首先将语言转化为正则表达式，为下一阶段的任务做准备

- **补点**

将原本缺失的点补上。比如原本 ab 的式子实质上是 a • b

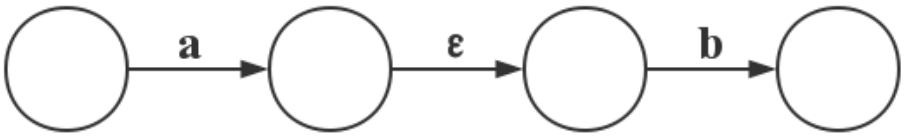
- **修改正则表达式**

将正则表达式修改为后缀形式，对于人观察而言，中缀表达式是比较

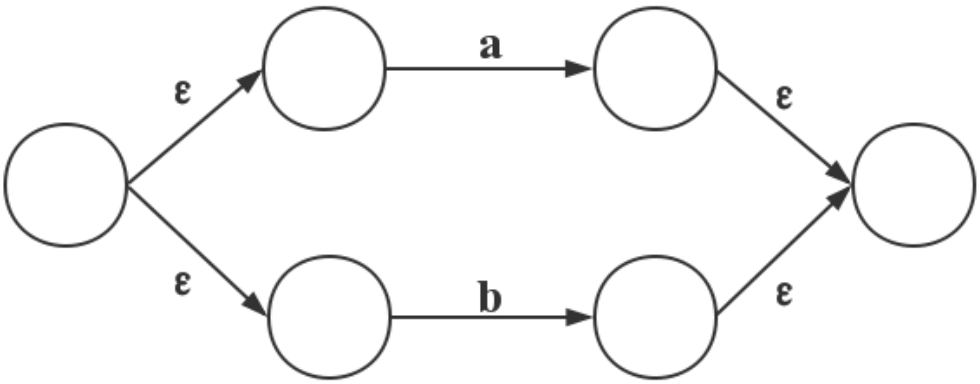
方便的，但是对于机器，后缀可以先观察到参与运算的操作数，然后看到操作符，对于机器处理更为方便。

- **构造 NFA**

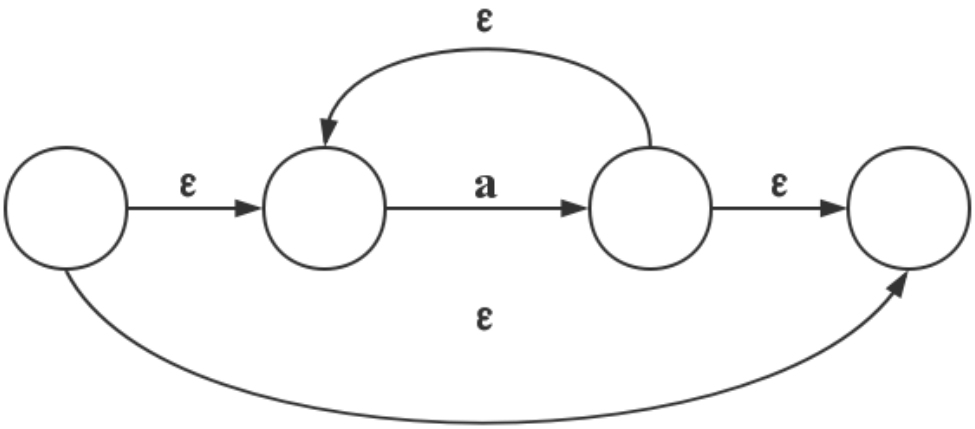
a • b



a|b



a*



三种小的情形，来将式的正则表达式修改为 NFA，为进一步构造 DFA 做准备

- **NFA 转化为 DFA**

使用子集构造法让构造得到的 DFA 的每个状态对于 NFA 的一个状态集合。DFA 在读入输入的字符之后到达的状态对应于相应的 NFA 从开始状态出发。

构造一个转化表，在表中每一个状态是一个 NFA 状态集合，我们将构造表，使得表可以并行的模拟输入一个给定的串时的可能的执行动作。首先，在读入第一个输入符号之前，N 可以位于集合的S_o闭包的任何状态上，其中S_o是初始状态。通过求出起始状态的通过不同边到达的状态的 ε 闭包构成新的状态，通过不断地迭代，直到没有新的状态再产生，就可以认为 DFA 所有的状态构造完毕。

- **DFA 转化为 DFA°**

使用自上而下的分类的方法将所有的状态按照以下规则进行划分。

强等价：两个状态的所有的发出边是相同的，且到达的状态也是相同的

弱等价：两个状态的所有的发出边是相同的，且到达的状态是属于相同的叶子节点

七、 实验心得

在本次实验中，通过个人使用 python，编写了 lex.py 实现了 C++词法分析器。在这次实验中，通过词法分析的具体实现过程，让我更加深切的体会到词法分析的过程中所涉及的各个具体的流程。本次实验中所采用的算法是 Tompson 构造法，在做完实验后也对该算法的流程有了一定的熟悉与更加深刻理解。本次实验中完成的词法分析器还不算完美， C++语言中的一些部分的运算符（主要是三元），小部分界符还是不能做很好的分析。但是可以分析绝大部分代码，还有相应的错误处理过程以及错误类型信息提示。日后要是还可以有机会，还会做更进一步的完善。