# 

编译原理实验

报告一

**实验名称： Lexical Analyzer**

**学生姓名： 梅洛瑜**

**学生学号： 71117408**

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 九 年 十 二 月



目录

[一、 实验目的 3](#_Toc2474)

[二、 实验环境 3](#_Toc27033)

[三、 实验内容 3](#_Toc7109)

[四、 实验结果 7](#_Toc1095)

[五、 数据结构 8](#_Toc11997)

[六、 实验算法 9](#_Toc7991)

[七、 实验心得 11](#_Toc26760)

1. 实验目的
   1. 在掌握RE工具包的基础上，使用汤普森完成了支持C语言的大部分文法识别
   2. 掌握有限自动机汤普森算法，理解处理输入的RE以及获得相应的token序列输出的过程
   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
2. 实验环境
3. 开发环境：
   * 操作系统：Windows 10
   * 源码：Python 3.7
   * 打包：Pyinstaller
4. 运行方式：
   * Windows图形界面
5. 实验内容

·在进行本次实验过程中，我首先使用RE工具包来完成了完整的C语言语法识别程序。此方法虽然实现了功能完善的词法分析器，但是没有使用汤普森算法，为了练习汤普森算法，我重新编写了基于FA的词法分析器并将两者结合了起来。使用汤普森算法识别C语言语法并支持错误抛出。

**·**实验要求实现能理解C的词法分析器，保证支持尽可能多的C语言语法。要求支持输入.c文件并能识别相应的四则运算，支持C语言中运算符、分隔符、数字、界符、变量名、方法名、注释的识别并在部分情况下能进行错误的识别与抛出

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。

* 分析正规文法

词法分析程序可以将RE正规文法转为DFA并保存至pkl**（pickle文件是python用于保存字符串、列表、字典等数据的典型文件）**文件中，在分析时读取相应的DFA并分析C语言文法。DFA存储方式见表.1.

Table. 1. 数据存储方式

|  |  |
| --- | --- |
| 文件名称 | 存储内容 |
| Start.pkl | 存储开始节点 |
| End.pkl | 存储终止节点 |
| DFA.pkl | 存储转换规则 |
| All.pkl | 存储终节点对应的词类型 |

* 词法分析

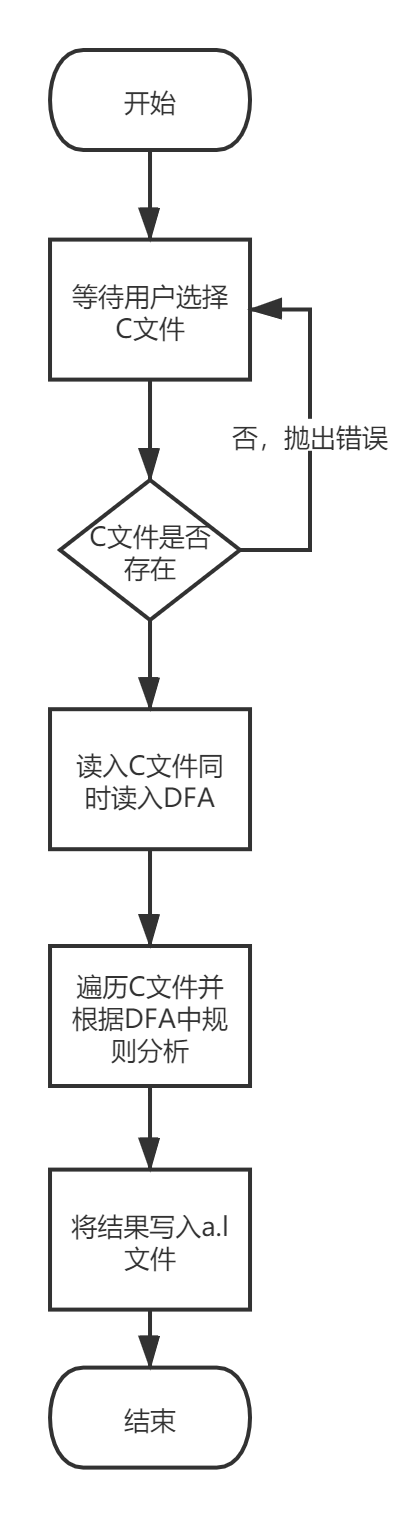
基于汤普森的词法分析器通过读取保存在pkl文件中的DFA，对读入的C文件字符序列进行遍历，按照DFA中的方法进行解释并设定内部码。在碰到我所设计的汤普森算法所不包含的构造规则时交由RE工具包进行处理。将我的算法与RE工具包结合以获得更加强大的词法分析器。

* 外部参数

本实验中的词法分析器提供从外部读入DFA的功能，更改pkl文件中的DFA将对分析器输出的token造成影响。同时，为方便用户操作提供直接读入C文件并输出token序列至.l文件的图形化界面。

程序流程图如图.1.

Fig. 1. 程序流程图



* 错误处理

支持的错误类型如表.2.（部分可能无法分析）

Table. 2. 错误类型

|  |  |
| --- | --- |
| 类型 | 描述 |
| 操作符错误 | 如：+\*，-+，/+ |
| 数字混有字母 | 123w |
| 小数点错误 | 如：1.1.2 |
| 标识符命名错误 | =w1 |
| 分号后出现语句 | ...; int 或 ...;\注释 |

设计的正规文法

* Operator

RE=(+-/& |\+=|\+|--|-=|-|\&=|/=|/|)【\*与正规表达式符号冲突采用&代替】

* Number

RE=(0-9)(0-9)\*

* Separator

RE=[,:\{}:)(<>]（分隔符/界符序列）

* Float

RE=(0-9)(0-9)\*.(0-9)(0-9)\*

* Keyword

RE= ['auto', 'struct', 'if', 'else', 'for', 'do', 'while', 'const',

'int', 'double', 'float', 'long', 'char', 'short', 'unsigned',

'switch', 'break', 'defalut', 'continue', 'return', 'void', 'static',

'auto', 'enum', 'register', 'typeof', 'volatile', 'union', 'extern']

* ID

RE= [a-zA-Z\_][a-zA-Z\_0-9]\*

* Method

RE= (main | printf)

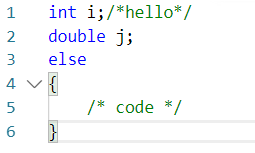
* Error

RE= (.\*\S+)

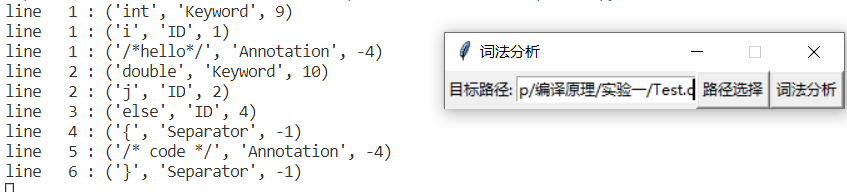
1. 实验结果

该实验结果由以下简短的C程序所构造得出。

**Test.c:**



**运行结果:**



**生成的a.l:**

line   1 :

[('int', 'Keyword', 9)]

[('i', 'ID', 1)]

[('/\*hello\*/', 'Annotation', -4)]

line   2 :

[('double', 'Keyword', 10)]

[('j', 'ID', 2)]

line   3 :

[('else', 'ID', 4)]

line   4 :

[('{', 'Separator', -1)]

line   5 :

[('/\* code \*/', 'Annotation', -4)]

line   6 :

[('}', 'Separator', -1)]

1. 数据结构

|  |  |  |
| --- | --- | --- |
| **名称** | **类型** | **描述** |
| **Operator** | Defaultdict(list) | 放置标识符 |
| **Separator** | Defaultdict(list) | 分隔符 |
| **Number** | Defaultdict(list) | 整数变量 |
| **Error** | Defaultdict(list) | 错误类型 |
| **Float** | Defaultdict(list) | 浮点数变量 |
| **Keyword** | Defaultdict(list) | 关键字 |
| **ID** | Defaultdict(list) | 标识符 |
| **Method** | Defaultdict(list) | 方法名 |
| **NFA** | Dict | NFA转换关系 |
| **DFA** | Dict | DFA转换关系 |
| **Queue** | Queue | 放待填表的状态 |
| **All** | Dict | 存放终节点和对应的词语类型 |
| **Stack** | Stack | 后缀表达式分析时使用 |

1. 实验算法

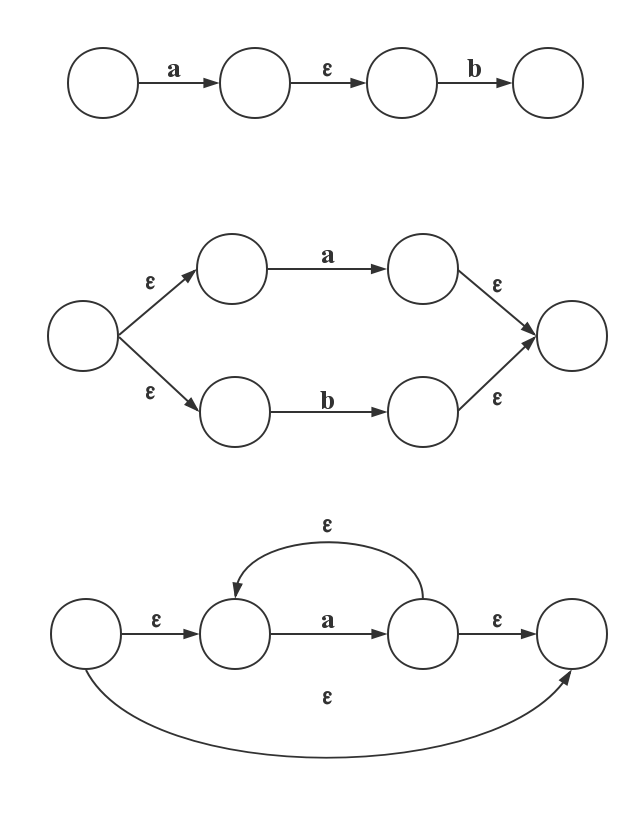
此次实验首先采用了RE工具包实现了完整的词法分析器，然后使用汤普森算法重新完成了词法分析的功能，并支持一定的错误检测。同时，由于我实现的算法对C语言无法完全支持，在遇到无法解析时再次调用RE工具包来实现对本算法的补充。

* **正则表达式**

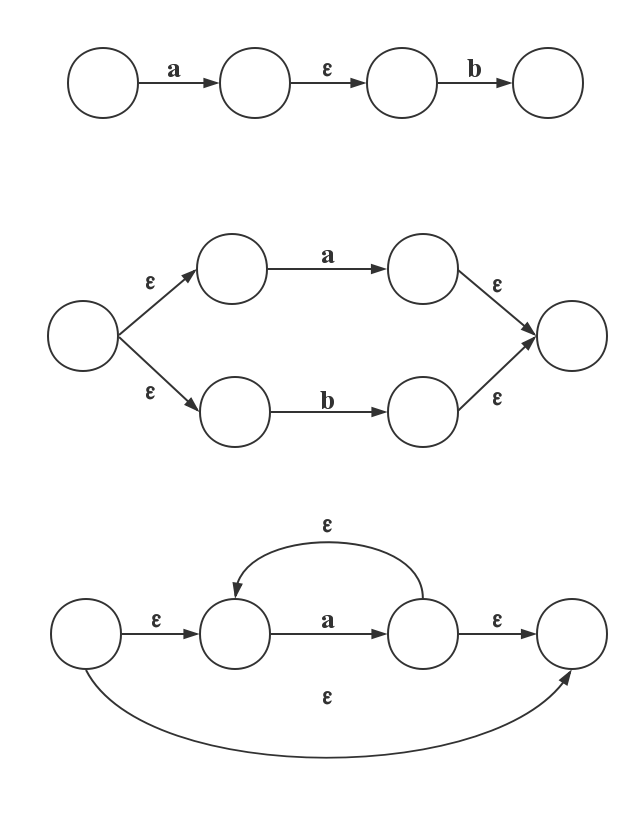
首先将语言转化为正则表达式，为下一阶段的任务做准备。由于机器采用从左到右扫描的方式来分析字符流，于是需要将C语言惯用的中缀表达式变成后缀表达式形式从而避免了回溯。

* **NFA**

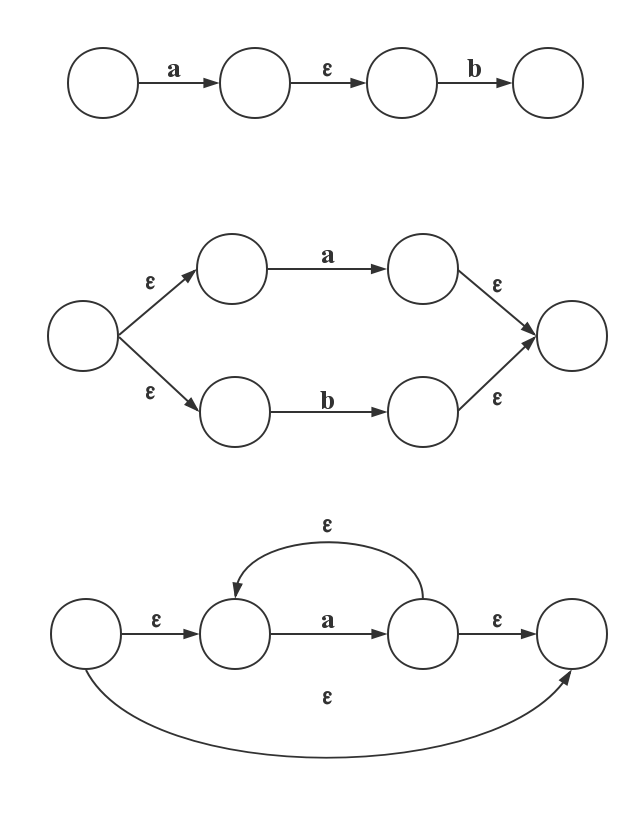
**a·b**



**a|b**



**a\***



三种小的自动机，来将式的正则表达式修改为NFA，为进一步构造DFA做准备。

算法实现:

用stack来对后缀表达式转换分析，用list表示节点间的转换关系。碰到符号则压入栈中，碰到·，、|、\*等字符则调整栈并加入相应的节点，合并成一个大的节点，压入栈中，最后变成一个节点，NFA构造就此完成。

* **NFA转化为DFA**

子集构造法来实现NFA转换为DFA。使用Dist键值对来保存生成的DFA的状态。首先构造开始状态的ε闭包以获得大的开始状态S0并压入字典中格式为：【0, S0】，开始迭代。迭代中对于每个状态，计算其对于每个非终结符其生成的可到达状态集合，若此集合不在字典中则放入，否则继续寻找直到不再有新的状态加入字典中，将包含原始NFA的截止状态作为DFA的截止状态。

算法实现:

程序中将n个NFA合并成一个DFA。先将所有开始节点之前用ε边再连一个开始节点，作为总的开始节点。然后使用队列，将开始节点放入队头，产生的状新态先填表再依次放入队列，产生的旧状态只填表。每次对队头的状态进行填表操作，直到队列为空，所有状态的填表操作完成。此时的状态转换字典表(类似哈希表，键值对为边和下一状态)即为DFA，终节点对应自己的词语类型，保留字会覆盖标识符。

* **DFA转化为DFA****°**

根据计算出的DFA状态，从所有状态组成的状态集作为根节点出发，对于转换不同（不满足弱等价）的状态节点分化为两个节点。迭代上述过程直到所有处在同一节点的状态都至少为弱等价（注意回头看问题）。

* **整体思路**

C语言程序——>RE文法（内嵌）——>NFA——>DFA——>Scanner

七、实验心得

此次实验，掌握了实现从正规文法构造NFA并用子集构造法建立DFA最后形成DFA°的过程。程序的最终成果为：读入C文件并形成相应的token序列，我通过结合汤普森算法与RE工具包，完成了大部分词法规则的识别，并利用python的tkinter包制作了简单的图形界面，支持选择C文件并将token序列输出至a.l文件中。通过此次实验，我对词法分析器的功能与实际实现有了清晰的认识，并且知道python中的字典是一种很适合储存状态与转换关系的数据结构，并且其由于使用了Hash匹配访问复杂度O(1)，这将会对实验二的语法分析数据结构设计有很大帮助。