编译原理实验报告

词法分析

北京理工大学计算机学院

谭超 1120161874

2019.4.19

实验目的：

（1） 熟悉 C 语言的词法规则，了解编译器词法分析器的主要功能
（2） 掌握典型词法分析器构造的相关技术和方法，设计并实现 C 语言词法分析器；
（3） 掌握编译器从前端到后端各个模块的工作原理，词法分析模块与其他模块之间的交互过程

实验内容：

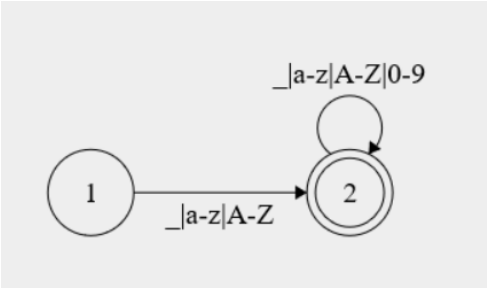
根据 C 语言的词法规则，设计识别 C 语言所有单词类的词法分析器的确定有限自动机，并使用 python 采用程序中心法或数据中心法实现词法分析器。词法分析器的输入为 C 语言源程序，输出为属性字流。

实验的具体过程和步骤：

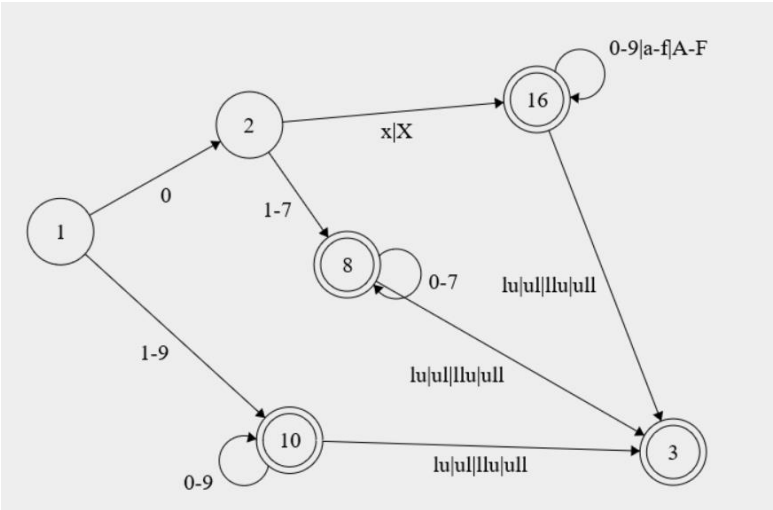
整个 C 语言的单词类大致分为 8 类：关键字、分隔符、运算符、标识符、整型常量、浮点型常量、字符常量、字符串常量。对于关键字、分隔符和运算符，由于符号确定，分别使用列表存储。对于其余的三类，使用有限自动机识别，其确定的有限自动机如下：

其中 character_set 为字符集，escape_sequence={\',\",\?,\\,\a,\b,\f,\n,\r,\t,\v}

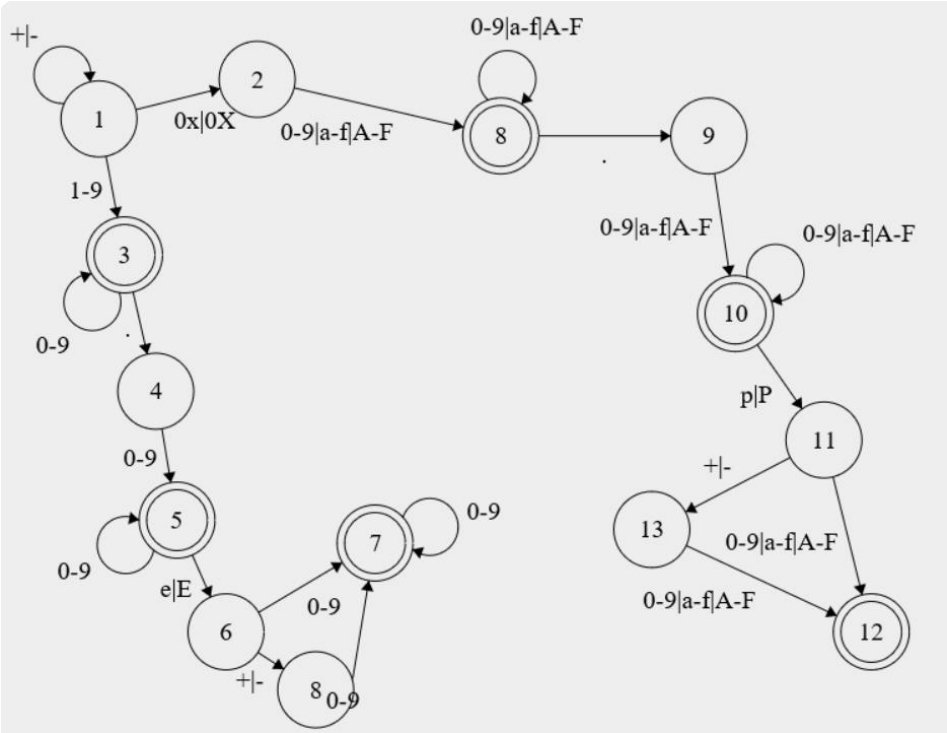
标识符：



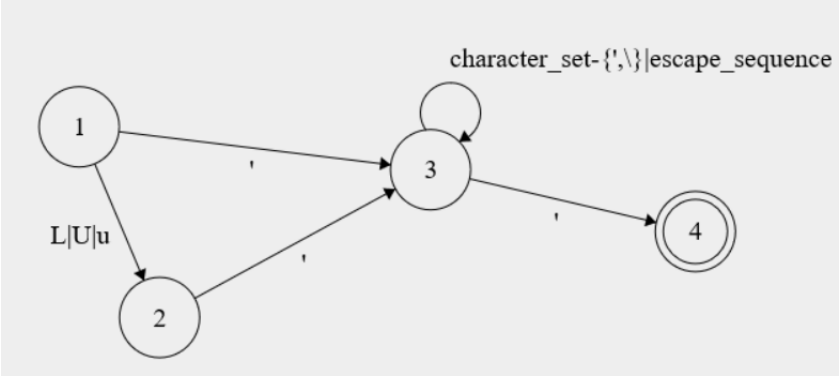
整型常量：注意 l、u 不区分大小写



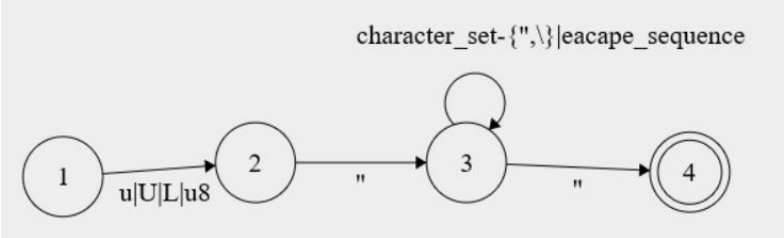
浮点型常量：



字符常量:



字符串常量:



然后根据确定的有限自动机来构建相应的单词类正则表达式，其构造结果如下:

```
keyword=['auto','break','case','char','const','continue','default',\
    'do','double','else','enum','extern','float','for','goto','if',\
    'inline','int','long','register','restrict','return','short',\
    'signed','sizeof','static','struct','switch','typedef',\
    'union','unsigned','void','volatile','while']
separator=['[',']','(',')','{','}','.','/',':',';','...',',','#',\
          '##','<:',':>','<%','%>','%:','%:%:']
operator=['->','++','--','&','*','+','-','~','!','%','<<','>>','<',\
        '>','<=','>=','==','!=','^','|','&&''||','?','=','*=','/=','%=',\
        '+=','-=','<<=','>>=','&=','^=','|=','']
identifier=re.compile(r'^[a-zA-Z\_][a-zA-Z0-9\_]*$')
integer_constant=re.compile(r'[+-]?[1-9][0-9]+|[+-]?0[0-7]+|[+-]?0[xX][0-9a-fA-F]+')
floating_constant_D=re.compile(r'[-+]?([0-9]+(\.[0-9]+)?|\.[0-9]+)([eE][-+]?[0-9]+)?')
floating_constant_H=re.compile\
    (r'[-+]?0[xX]([0-9a-fA-F]+(\.[0-9a-fA-F]+)?|\.[0-9a-fA-F]+)([pP][-+]?[0-9a-fA-F]+)?')
character_constant=re.compile(r'^\'[^\'\\]\'&')
string_literal=re.compile(r'^\"[^\'\\]+\"&')
```

首先创建一个 dom 对象树，在该对象树下创建一个子节点名为 project，再在 project 节点下创建一个子节点 tokens 用来存放单词。从经过预处理后的文件中读取单词，然后判断该单词是否一个关键字、分隔符或运算符，如果都不是，与已建立的正则表达式进行匹配，如果匹配成功，则生成一个子节点 token，将序号、值、类型、行号、是否有效写入到 token 中，然后将 token 压入 tokens 节点，如果匹配都不成功，创建子节点 token，将该单词相关信息写入其中，并置 valid=false,type=unknow。待所有单词处理完毕，调用 writexml()将 dom 的内容写入到文件中去。

至此，词法分析器已经完成，接下来便是如何将该词法分析器嵌入到 BitMiniCC 框架中去：首先通过 pyinstaller 将 python 程序生成 exe 程序：

```
PS D:\Github\my repository\bit-minic-compiler-master\myScanner> pyinstaller -F .\myScanner.py
94 INFO: PyInstaller: 3.4
95 INFO: Python: 3.7.0
95 INFO: Platform: Windows-10-10.0.17134-SP0
97 INFO: wrote D:\Github\my repository\bit-minic-compiler-master\myScanner\myScanner.spec
99 INFO: UPX is not available.
100 INFO: Extending PYTHONPATH with paths
['D:\\Github\\my repository\\bit-minic-compiler-master\\myScanner',
```

然后将生成的 exe 程序放到 BitMiniCC 框架的 bin\binary 目录下，并修改配置文件 config.xml：

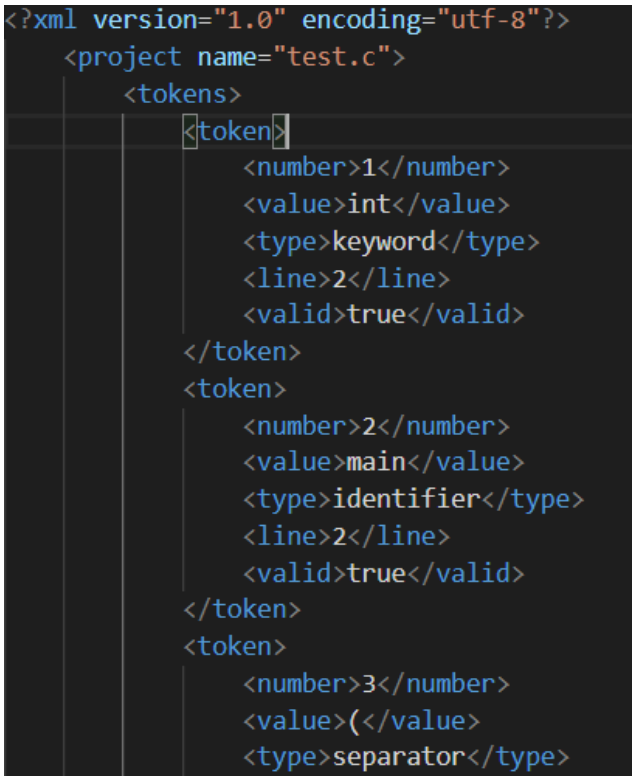
```
<?xml version="1.0" encoding="UTF-8"?>
<config name="config.xml">
  <phases>
    <phase>
      <phase skip="false" type="java" path="" name="pp" />
      <phase skip="false" type="binary" path="D:\Github\my repository\bit-minic-compiler-master\bin\binary\myScanner.exe" name="scanning" />
      <phase skip="false" type="java" path="" name="parsing" />
      <phase skip="false" type="java" path="" name="semantic" />
      <phase skip="false" type="java" path="" name="icgen" />
      <phase skip="false" type="java" path="" name="optimizing" />
      <phase skip="false" type="java" path="" name="codegen" />
      <phase skip="false" type="java" path="" name="simulating" />
    </phase>
  </phases>
</config>
```

配置已经完成，接下来就可以通过/bin/binary/run.bat 来运行整个框架：

```
PS D:\Github\my repository\bit-minic-compiler-master\bin> .\run.bat ..\input\test.c

D:\Github\my repository\bit-minic-compiler-master\bin>java -jar ..\lib\BITMiniCC-obf.jar ..\input\test.c
Start to compile ...
1. PreProcess finished!
2.myScanner finished!
```

运行效果截图：



实验心得体会：

在词法分析器的构建过程中，有限自动机的作用是不可忽视的，而通过确定的有限自动机，我们可以轻松的写出其对应的正则表达式，从而实现单词类的识别。利用正则表达式实现和通过程序中心法实现的词法分析器相比较无疑要简单的多，但同时也有一些不足，最突出的就是如果匹配出错，如果利用正则表达式匹配则只能告诉用户该单词词法有问题，而不能指出是什么问题而通过程序中心法实现的话可以根据当前状态和输入来大致的判断并告诉用户哪里出错了以及出的什么错。在将词法分析器嵌入框架的时候需要在配置文件中设置路径，此处只能设置绝对路径而无法设置相对路径，然而，由于程序可能在在其他电脑上运行，所以在其他电脑上运行的时候会出错，目前我也没有找到解决的方法。