****

词法分析实验

班 级： 07111505 \_

姓 名： 王阁元

学 号： 1120151858\_

目录

[一、 实验目的 1](#_Toc511144884)

[二、 实验内容 1](#_Toc511144885)

[三、 实验环境 1](#_Toc511144886)

[四、 实验过程及具体步骤 1](#_Toc511144887)

[4.1 明确实验要求 1](#_Toc511144888)

[4.2 明确实验框架 2](#_Toc511144889)

[4.3 实现识别关键字和标识符的自动机 3](#_Toc511144890)

[4.4 实现识别分隔符的自动机 3](#_Toc511144891)

[4.5 实现识别操作符的自动机 3](#_Toc511144892)

[4.6 实现识别常量的自动机 4](#_Toc511144893)

[4.7 最终的词法分析的自动机 6](#_Toc511144894)

[4.8程序实现 8](#_Toc511144895)

[4.8 总结 8](#_Toc511144896)

[五、 运行结果截图 9](#_Toc511144897)

[六、 心得体会 11](#_Toc511144898)

## 实验目的

在已有的BITMiniCC框架的基础上，实现词法分析器，加深对词法分析的理解。实现对输入C语言源文件进行词法分析得到属性字流。

## 实验内容

以C语言为源语言，java语言为宿主语言，构建C语言的词法分析器，对于任一给定C语言源程序，输出XML格式的属性字流。

## 实验环境

操作系统：Windows10

IDE：Eclipse

JDK版本：1.8.0

## 实验过程及具体步骤

### 4.1 明确实验要求

本实验中的C语言词法分析器基于C99标准实现。其中C语言的词法元素分为5大类：关键字、标识符、操作符、分隔符、常量。其中常量包括八进制、十进制、十六进制常量，科学记数法，字符、字符串常量。本实验实现了对常见C语言语法的正确的词法分析，对于一些常见的错误也会进行出错处理，并且实现了对每个单词所在行号的记录。

本实验实现的要求如下：

●关键词

●标识符

●操作符：“+”，“-”，“++”，“--”，“+=”，“-=”，“\*”，“/”，“%”，“^”，“=”，“！”，“\*=”，“/=”，“%=”，“^=”，“==”，“!=”，“&”，“&&”，“&=”，“|”，“|=”，“||”，“<”,“<=”,“<<”,“<<=”,“>”,“>=”,“>>”,“>>=”，“~”

●分隔符：{}[](),;.:?

●八进制、十进制、十六进制整数和浮点数（负数的负号视为操作符，其具体含义应放在语法分析部分，再此未将负数视为一个单词）

●科学记数法

●字符、字符串常量（包括转义符）

●记录每个单词出现的行数

●部分非法字符的处理。

### 4.2 明确实验框架

由于本次实验我选择的宿主语言为java语言，因此，在实验过程中我采用面向对象的思想，设计了三个类：Scanner, Token, LexAnalyse。具体设计如下：

#### 4.2.1 Scanner

用于生成LexAnalyse对象，然后调用词法分析类的构造函数。

#### 4.2.2 Token

记录单个单词分析得到的结果。

主要成分为：单词序号，单词的值，单词类型，单词所在行号，单词是否合法。

提前将分隔符和关键字存储于列表中，方便对关键字和分隔符进行判断。

#### 4.2.3 LexAnalyse

主要成分为：文件I/O对象，当前行数，当前读文件读到的位置，存储单词的列表。

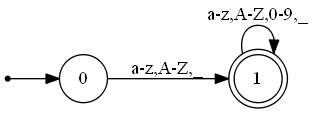
主要实现方法：读写文件，自动机状态转换。

实现对C语言源代码的词法分析。包括以下部分：

### 4.3 实现识别关键字和标识符的自动机

根据标识符的定义：由字母、下划线、数字组成且开头不能为数字的单词。

下图为识别标识符的自动机。



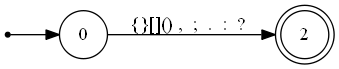
对于关键字的判断，只需每次对已经识别的标识符进行判断看其是否为关键字即可。

### 4.4 实现识别分隔符的自动机

C语言中常见的分隔符有：{}[](),;.:?

分隔符为单个字符，较容易识别。

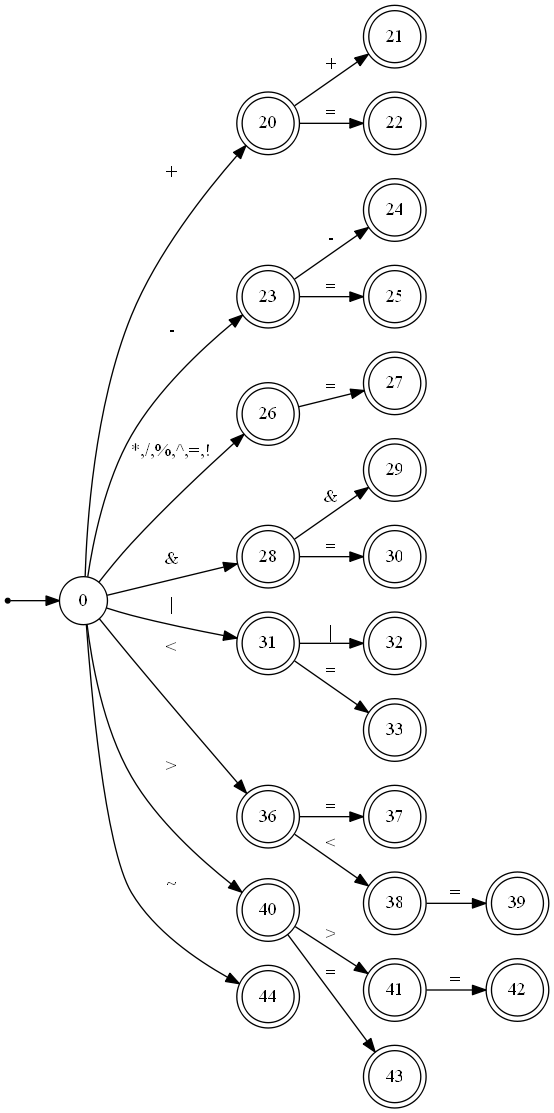
下图为识别分隔符的自动机



### 4.5 实现识别操作符的自动机

常见的操作符有：“+”，“-”，“++”，“--”，“+=”，“-=”，“\*”，“/”，“%”，“^”，“=”，“！”，“\*=”，“/=”，“%=”，“^=”，“==”，“!=”，“&”，“&&”，“&=”，“|”，“|=”，“||”，“<”,“<=”,“<<”,“<<=”,“>”,“>=”,“>>”,“>>=”，“~”。

下图为识别操作符的自动机：



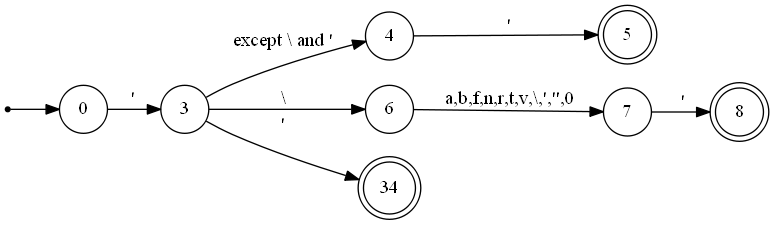
### 4.6 实现识别常量的自动机

常量的识别较为复杂。包括八进制、十进制、十六进制整数和浮点数、科学记数法、字符、字符串常量（包括转义符）。

#### 4.6.1 识别字符常量

字符常量包括普通字符和转义字符。字符常量都是由一个单引号开始，中间为一个普通字符或者转义字符（或者为空），最后由一个单引号结束。

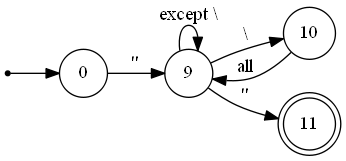
下图为识别字符常量的DFA：



#### 4.6.2 识别字符串常量

字符串常量可以为空，也可以是字符的连接，其中可以包括转义字符。

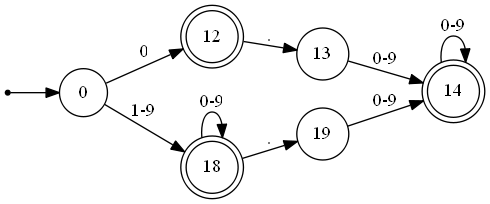
下图为识别字符串常量的DFA：



#### 4.6.3 识别十进制整数和小数

十进制整数和小数，对于首位，如果是0，那么一定为小数；否则将会发生错误。对于后边的位数，均可以是0-9。

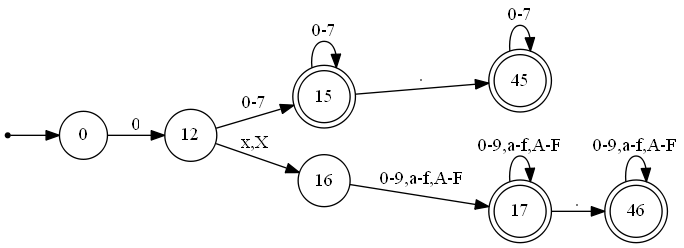
下图为识别十进制整数和小数的DFA。



#### 识别8进制和16进制整数和小数

下边考虑了8进制和16进制整数和小数。注意每位的取值范围即可。另外他们也存在小数。

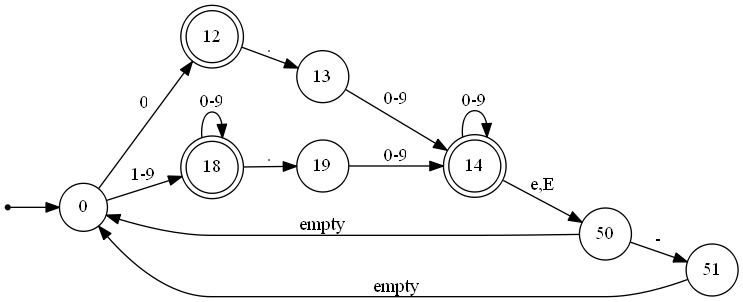
下图位识别8进制和16进制整数和小数的自动机。



#### 识别科学记数法

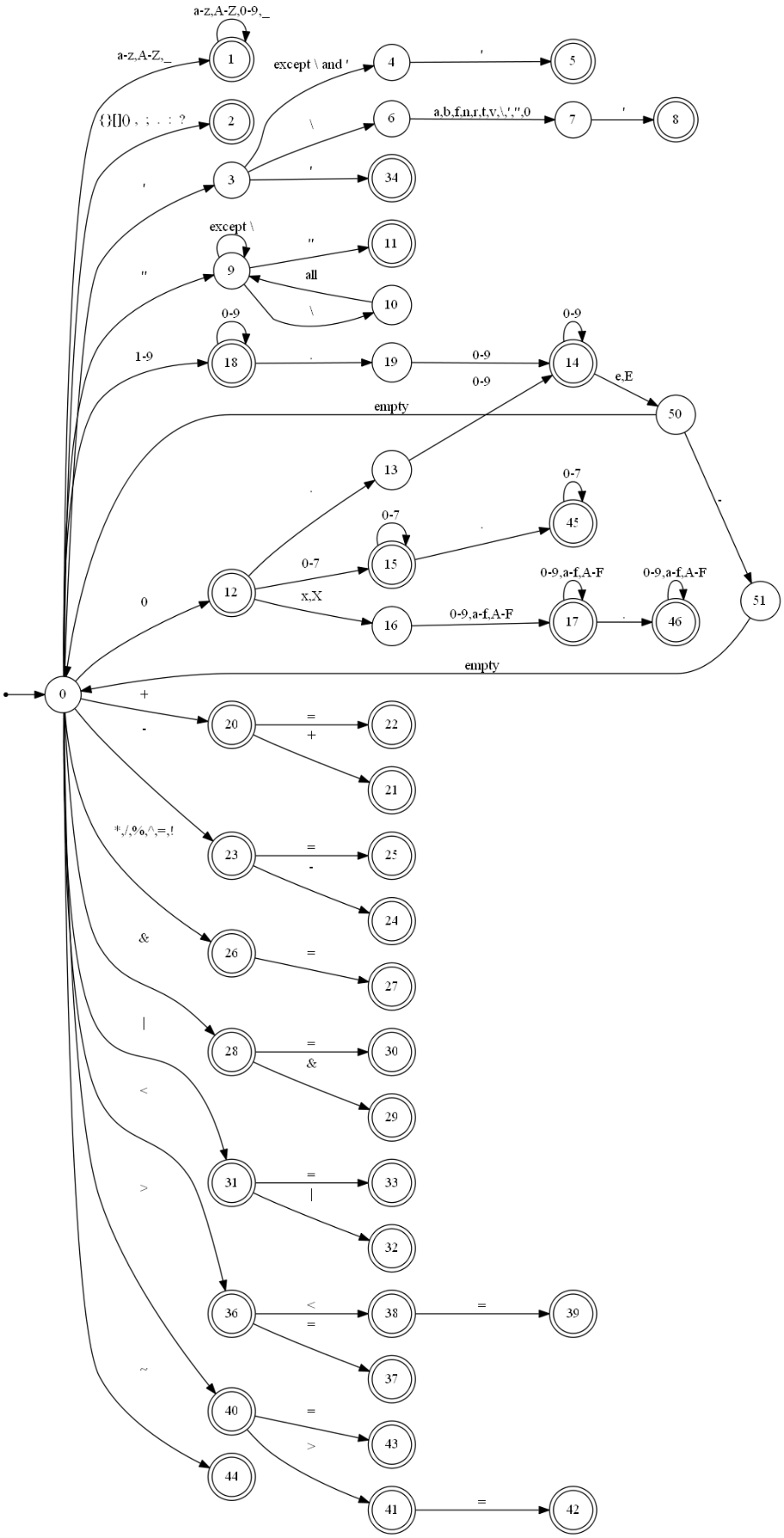
科学记数法的表示形式为：1e10或者1.1E-2.1。由于科学记数法的指数部分和系数为一体，因此如果指数为负数，那么此处的负号不是操作符，而归于科学记数的这个类别。

下图为识别科学记数法的自动机：



### 最终的词法分析的自动机

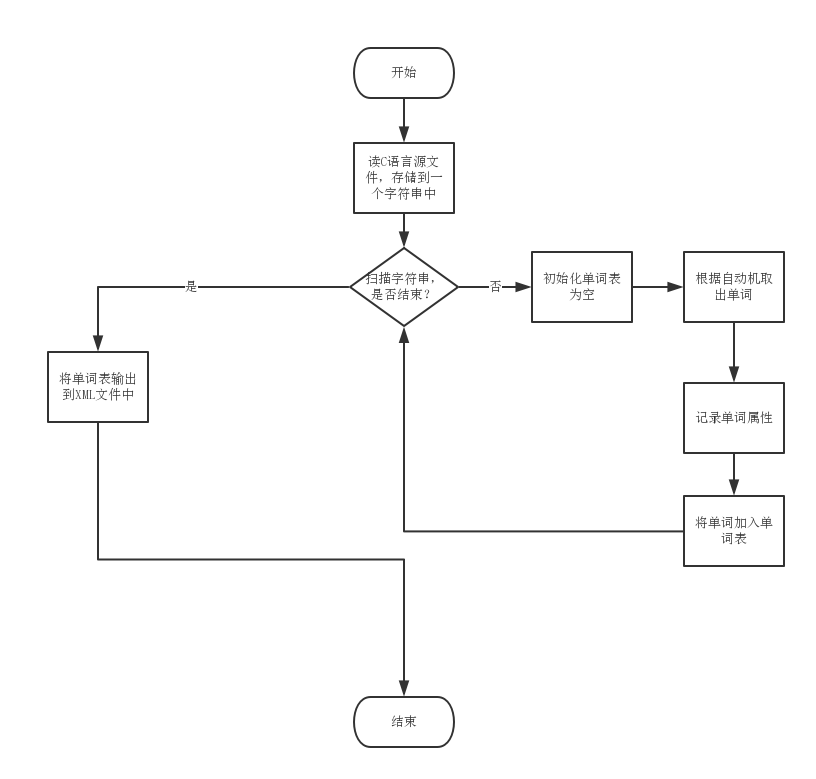
将前边的DFA各个分量进行汇总，合并成一个最终的具有完整功能的DFA。



### 4.8程序实现

程序使用Java实现，使用以程序为中心的方法，对于上边各个分自动机定义一个状态转移函数，以提取每一个单词。对每一个单词标注编号、行号、值、类型、是否有效等。

下边为程序框图：



### 总结

本实验实验的词法分析器实现了如下详细功能：  
●关键字和标识符的支持

●运算符和分隔符的支持

●数字常量的支持：十进制整数，八进制整数，十六进制整数，十进制浮点数，八进制浮点数，十六进制浮点数，科学计算法支持。

●字符类型和字符串类型支持（包括转义字符）

●支持行数统计

●支持部分的出错处理

## 运行结果截图

本部分是基于自己的项目运行得到的结果，不过由于框架预处理有错误，会对本样例预处理后得到的结果直接转成空文件，导致接下来的步骤无法进行。

源文件（.c文件）：

|  |
| --- |
| void function(int a){  int b=0x2af;  int c=a++ + ++b+b+++a--;  }  typedef struct note {  int c;  }NODE;  enum t {  e1=1,  e2=2,  e3=3,  e4=4  };  int main(int argc, char\* argv[])  {  a=-2012.34E-301;  b=+2.13;  NODE r;  int a,b;  char\* w=NULL;  int s[10]={1,2,3,4};  char\* pp="a string\n";  int p=0.001;  \*w='a';  w=&w;  a+++b;  b/=a;  r.c;  a+b/2,2-1;  int\* c;  \*c=1;a=1;p=2;  a=++p;  switch (a){  case 1: a=1;break;  case 2:b=1;break;  default:printf("something wrong!\n");break;  }  c=a>b?a:b;  c!=a;  if(a!=23){  c=a;b%=3;  }else{  function(23);  }  for(int i=1;i<10;i++){  printf("hello %d\n");  }  while(a&&1){  b=a^1;  c^=2;  d>>=1;  }  return 0;  } |

词法分析结果的XML文件（文件太大，只显示了一部分）：

|  |
| --- |
|  |

## 心得体会

通过本次实验，我掌握了词法分析器的核心技术，对于复杂自动机的实现有了很大进步。最大的体会就是，java在面向对象的实现方面还是相当的规矩，不容易出错。在复杂自动机的实现上，整体的把握相当重要，由于一开始很多公共的操作，我没有封装到一个函数中，导致了代码的少量重复。另外，在自动机的不断完善过程中，可能会出现牵一发而动全身的现象，最后我也是反复改进初始状态的一些转移才得以消除错误。

其次，就是自动机的设计过程中，一开始我直接计划将完整自动机画出来之后再进行程序实现，但是由于完整的自动机需要考虑太多方面，于是最后还是选择了分为五大部分进行实现。后来事实证明我的选择是正确的，分模块会让整个结构显得更加明确，对于单个模块的出错处理也是更加清晰。

最后谈一下方法，我才用的是以程序为中心的方法，由于之前对出错处理考虑的不是很全面，导致最后出现出错处理需要修改大量起初代码，由于时间关系，我只实现了一些容易添加的错误处理，但并不完整。这也让我在以后的设计中有了反思，一定要设计好整体的框架，然后实现，这样才不至于出现实现过程中发现问题，无法更改的惨状。