一、 实验目的和内容

1. 实验目的

（1）熟悉 C 语言的词法规则，了解编译器语法分析器的主要功能；

（2）熟练掌握典型语法分析器构造的相关技术和方法，设计并实现具有一定分析能力的 C 语言语法分析器；

（3）掌握编译器从前端到后端各个模块的工作原理，语法分析模块与其他模块之间的交互过程

2. 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状态自动机，并使用 Java、C\C++或者 Python 其中任何一种语言，采用程序中心法或者数据中心法设计并实现词法分析器。词法分析器的输入为 C 语言源程序，输出为属性字流

二、 实现的具体过程和步骤

1. 对词法进行划分

词法的识别采用自动机来进行，一般情况下，都是为一类词法构建一个自动机，例如，有识别数字的自动机、有识别字符串的自动机。词法的划分是用 DFA 进行识别的基础，也可以理解为将识别不同单词的自动机整合为一个巨型自动机的前提。只有在对当前输入进行判断，选择正确的自动机以后，才能正确的进行词法分析。

C 语言中词法规则分为 7 类，分别是关键字，标识符，整形常量，浮点常量，字符串常量，字符常量，运算符与界限符。程序中包含所有的 7 类规则，但是初次选择只划分为 5 类，在把相关字符串从文件中读取出来以后，再进行详细划分。

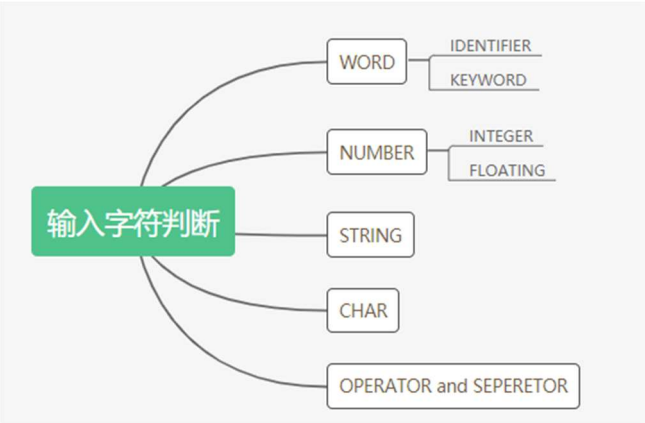

Figure 1 词法划分

核心代码如下所示：

```
1. private void scanAll() throws Exception{
2.                 while(readChar()!=EOF){
3.                         if(isAlpha(ch)){
4.                                 String word= readWord();
5.                         }
6.                         else if(isNumber(ch)){
7.                                 String number=readNumber();
8.                         }
9.                         else if(ch=='\"') {
10.                                String s = readString();
11.                        }
12.                        else if(ch=='\''){
13.                                String char_constant=readCharCon
   stant();
14.                        }
15.
16.                        switch (ch) {
17.                            //operator and seperator here
18.                        }
19.  }
```

2. Token

Token 定义如下所示，相比于示例输出多出了行列以及合法性信息。行列信息以单词第一个字母所在位置为标准。合法性信息检测单词是否合法，在一些情况下读取错误的单词是很有必要的，例如“3m“不是一个合法的标识符，但程序仍将其识别为一个标识符，而不是数字 3 与标识符 m，最后将合法性属性设置为 false。

```
class  Token{
                int  number;
                int  row,col;
                String  type;
                String  value;
                boolean  valid;
}
```

3. 识别标识符和关键字

标识符和关键字的规则按照 C 标准。

在初步划分中，它们被归为一类，由相同的自动机识别。DFA 的规则很简单，读取数字、字母或者下划线直到不合法的输入。之后根据 C 语言规则判断单词的合法性，即：以下划线或者字母开始，之后可以是数字、字母或者下划线。

识别通过后，在关键字表中查询是否为关键字，并生成相应 Token。

标识符：

identifier → *identifier-nondigit*
| *identifier identifier-nondigit*
| *identifier digit*
identifier-nondigit → *nodigit*
| *universal-character-name*
| *other implementation-define characters*

4. 数字的识别

在 C 语言中数字常量分为两类，整型与浮点型，并且有不同进制的表示。程序实现十进制下整形与浮点型的识别，规则按照 C 标准。

无论是浮点数还是整型，都以数字开始，并且整型数是浮点数的前缀。因此首先读取数字直到不合法输入。之后进行浮点数的判断。

若是浮点数，根据规则，输入应为“.”开头的尾数部分或者“e”,“E”开头的指数部分。尾数部分仍然读取数字，指数部分则先读取符号，再读取数字。

根据这三个部分的读取结果可以判断输入是整型还是浮点型。之后进行合法性判断阶段，包括整数除 0 以外不能以 0 为起始数字，浮点数是否符合相应规定等。

整形：

integer-constant → *decimal-constant*
decimal-constant → *nonzero-digit* | *decimal-constant digit*
nonzero-digit → *1*|*2*|*3*|*4*|*5*|*6*|*7*|*8*|*9*

浮点型：

gloating-constant → *decimal-floating-constant*
decimal-floating-constant → *fractional-constant exponent-part*
| *digit-sequence exponent-part*
fractional-constant → *digit-sequence* . *digit-sequence* | *digit-sequence*
exponent-part → *e sign digit-sequence* | *E sign digit-sequence*
sign → + | -
digit-sequence → *digit* | *digit-sequence*

5. 字符串的识别

字符串规则按照 C 标准，省略字符串前缀。

输入字符为双引号即可判断为字符串输入，之后读取除了换行符的所有字符，直到下一个双引号。

需要注意的是转义字符的问题。简单的处理方式是，碰到转义字符则强制读取下一个字符，这样可以避免转义双引号带来的错误提前结束（示例的方式）。但是程序并没有采用这种方式，由于最终识别的结果是一个字符串，所以可以将其进行转义以后再加入字符串中，即将用两个字符标识的转义提前进行，变成一个字符，方便后续的处理。

实现的字符串常量规则如下：

string-literal → *“ s-char-sequence”*
s-char-sequence → *s-char* | *s-char-sequence s-char*
s → *any member of the source character set except the double-quote “, backslash \, or newline character* | *escape-sequence*

6. 运算符和界限符

按照标准实现了绝大多数词法，省略了不常见运算符。

运算符生成 Token 时也需要进行对下一个甚至下下个输入进行判断，之后才能确定运算符的真正类型。与示例不同的是，Token.type 属性的设置更为详细，而不只是简单的 OPERATOR，方便后续的处理。

实现的运算符和界限符如下：

```
[   ]   (   )   {   }   .    ->  ++  --  &   *   +   -   ~
    !   /   %   <<  >>  <    >   <=  >=  ==  !=  ^   |
    &&  ||  ;   =   *=  /=   %=  +=  -=  <<=     >>=
    &=  ^=  |=  ,
```

三、 运行效果截图

以 input/test.c 为例，展示赋值语句和函数调用语句形成的 Token 序列。详细的输出可以至 bin 目录里查看。

Figure 2 赋值语句

Figure 3 函数调用

四、 实验心得体会

词法分析实际上就是字符串处理的问题，但若只是盲目的进行处理很容易就会条理不清，从自动机的角度思考问题可以让逻辑与框架更为清晰。

字符串处理直接从文件进行，没有预先读入内存进行缓存，也就意味着读取过程不能回溯。但是在一些情况下需要对后续的字符进行判断才能处理，因此在第一个字符时就应考虑后续的可能。为此，程序预读一个字符，以实现对后续可能的判断。

Token 中增加了行列信息，这样在的错误处理时可以反馈精确的位置。这个信息由两个全局变量标识，为了正确的维护它们的值，程序中对读文件操作进行了封装，以函数调用方式读取文件就可以保证正确的位置信息。

还增加了正确性的信息。程序设计中很容易陷入的一个误区是，只对符合规则的输入进行读取，遇到不合法的就停止进行判断。但实际上程序应该像自动机一样，可以允许错误的输入。遇到错误的单词也应该生成相应 Token 而不影响后续的判断，即尽量将错误限制在一个单词中。