**编译原理与设计实验报告**

**姓名/学号**： 鲍吴迪/1120173588

**班级**： 07111704

# 实验名称

词法分析实验

# 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和实现技 术，掌握典型词法分析器构造方法，设计并实现 C 语言词法分析器;

(2)了解 Flex 工作原理和基本思想，学习使用工具自动生成词法分析器;

(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。

学生可以选择编码实现词法分析器，也可以选择使用 Flex 自动生成词法分 析器。需要注意的是，Flex 生成的是 C 为实现语言的词法分析器，如果需要生 成 Java 为实现语言的词法分析器，可以尝试 JFlex 或者 ANTLR。

由于框架是基于 Java 语言实现的，并且提供了相应的示例程序，建议学生 使用 Java 语言在示例的基础上完成词法分析器。

# 实验环境

Macbook pro

2.6Hz 双核Intel Core i5 内存8GB

Mac OS Catalina 10.15.1（clang/llvm编译器）

Intellij IDEA 2018.3.1

JDK 1.8.0

# 实验设计与实现

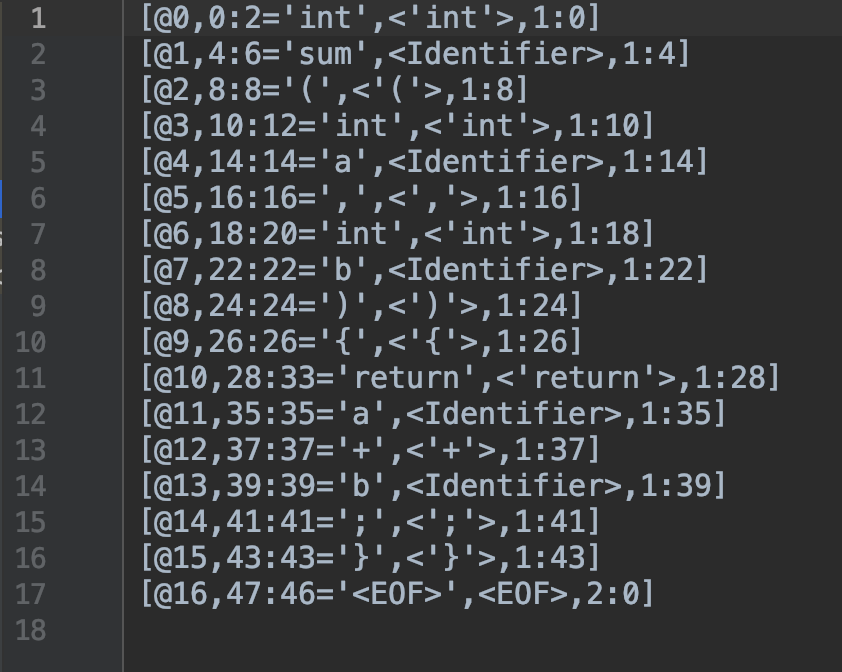
## 测试内置词法分析器，分析输出

将BIT-MINICC框架导入IntelliJ IDEA中，首先将JDK换为本地JDK，然后将config文件中的词法分析器、语法分析器path置空，使其使用内置词法分析和语法分析器，再将预处理跳过，以防止出现不必要的bug。

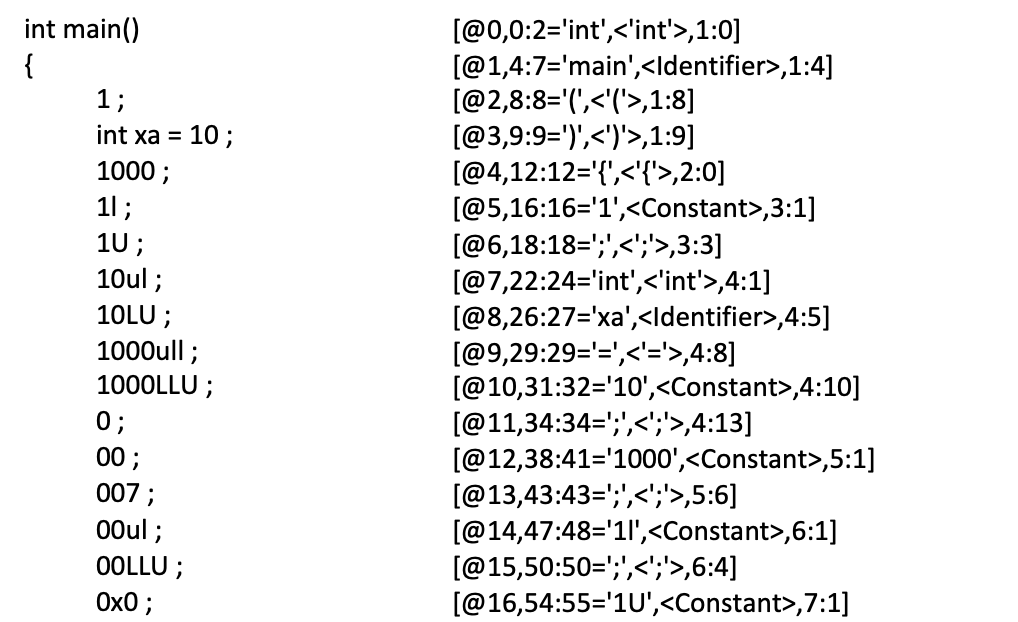
将scanner\_example.c文件路径设为BITMiNiCC.class的命令行参数后运行：

int sum(int a, int b) {  
 return a + b; }

查看输出结果：



将给出的textFile.c文件路径设为BITMiNiCC.class的命令行参数后运行，并将源码与testFile.token文件进行对比（仅截取一部分）：



可以看出，词法分析器输出格式为每行为一个 token，以@开头的数字表示 token 的 序号，紧接着的 xx:xx 表示 token 文本对应的开始列和结束列，“=”后面给出了 这个范围之内 token 的具体文本，“<>”之内表示 token 的类型，最后一个数字 对 xx:xx 表示起始行和起始列。需要说明的是，在这个例子中，运算符等的类型 就是其自身，属性流的最后放置了一个“EOF”表示属性字的结束位置。

需要注意的是：

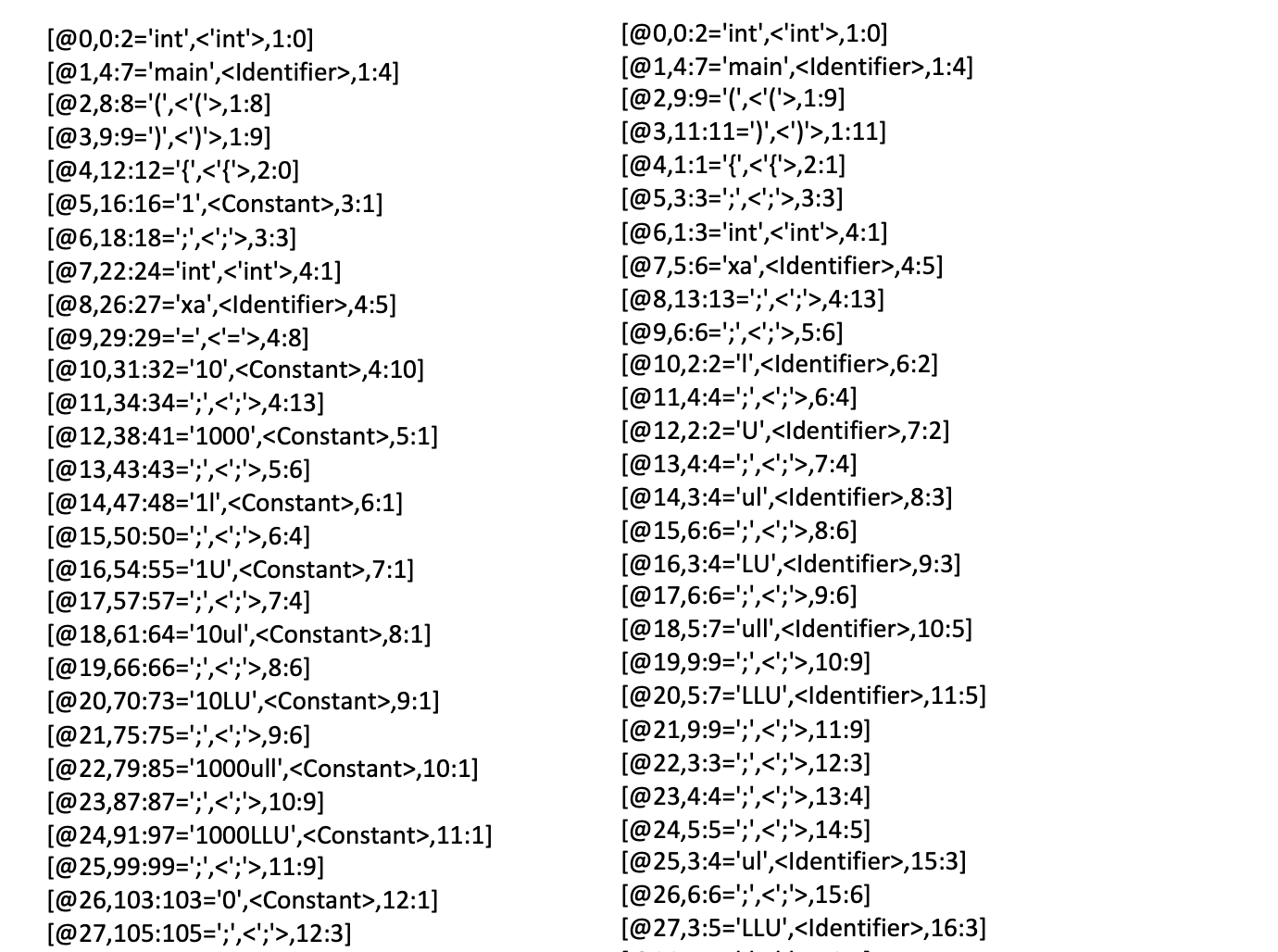
1. 每一个token的列号开始索引为从源码第一个字符一直到该token的第一个字符的累加，而给出的ExampleScanner中显示的是每一行的第一个字符一直到该token第一个字符的累加。
2. 对于字符常量识别为Constant，前缀为L,u,U,或无前缀，字符序列用’’包围。
3. 对于字符串面常量识别为StringLiteral，前缀为u8,u,U,L,或无前缀，字符序列用””包围。
4. 到达源文件末尾输出EOF：

[@435,1488:1488='}',<'}'>,129:0]

[@436,1489:1488='<EOF>',<EOF>,129:1]

行数不变，列数+1，token长度为-1（索引从1489:1488）。

## 运行示例词法分析器，分析输出结果



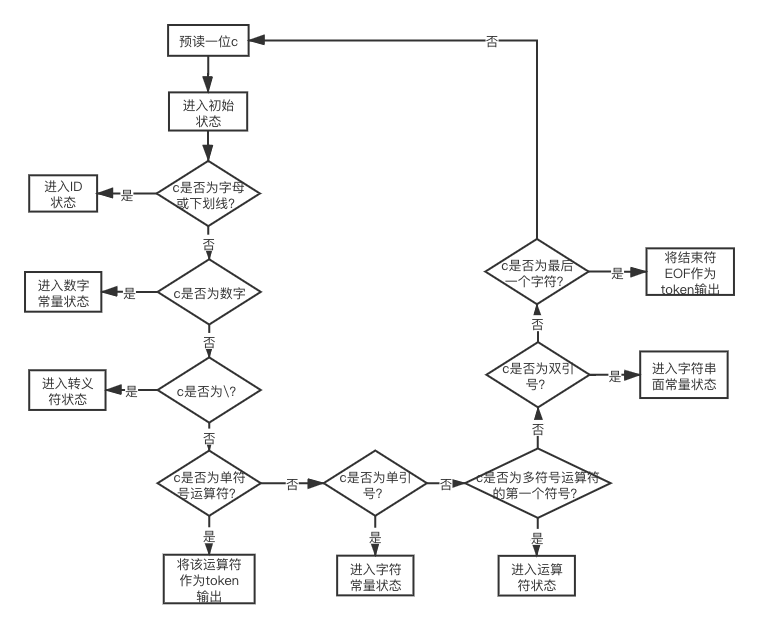
（左为内置词法分析器，右为示例词法分析器）

经观察，示例词法分析器存在以下不足：

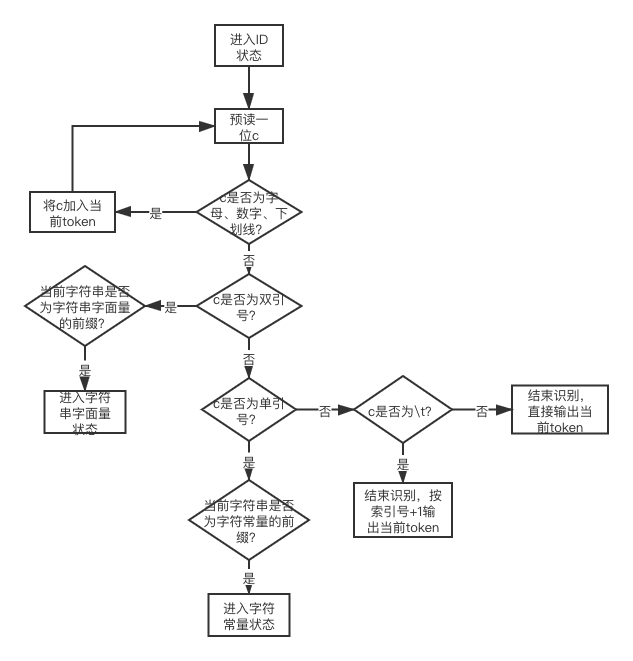
1. 关键字集不完整，许多关键字如static,sizeof等都直接识别为标识符。
2. 无法识别数字常量、字符常量、字符串面常量等一系列constant。
3. 无法识别以下划线\_开头的identifier。
4. 运算符集不完整，如无法识别\*，/运算符等。
5. token索引中记录的是在当前行的列数，而不是在整个源文件中的字符数累加。
6. 对于制表符\t及换行符没有进行索引的累加。
7. 文件末尾EOF关键字输出格式不正确。

## 实现词法分析器MyScanner，使其输出与内置词法分析器一致

部分较复杂状态算法流程图：



初始状态流程图



ID状态流程图

在scanner根目录创建立Myscanner.java文件，基础代码框架沿用ExampleScanner，针对示例词法分析器的不足，做以下改动：

1. 将DFA\_STATE中添加

DFA\_STATE\_CONS,//数字常量  
DFA\_STATE\_CONS2,//转义符号  
DFA\_STATE\_CONS3,//字符常量  
DFA\_STATE\_STRL,//字符串面常量  
DFA\_STATE\_CALC,//运算符

并在识别当前状态时为新加的状态编写相应转换或输出。

1. 在行计数器lIndex,列计数器cIndex之外添加字符计数器tIndex，用于表示当前字符在整个源码文件中的字符索引。其累加规则为：

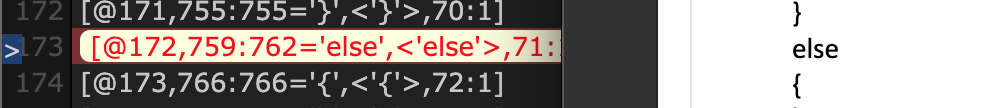
行数不改变时和cIndex一起+1；

行数改变时cIndex清零，tIndex+2；

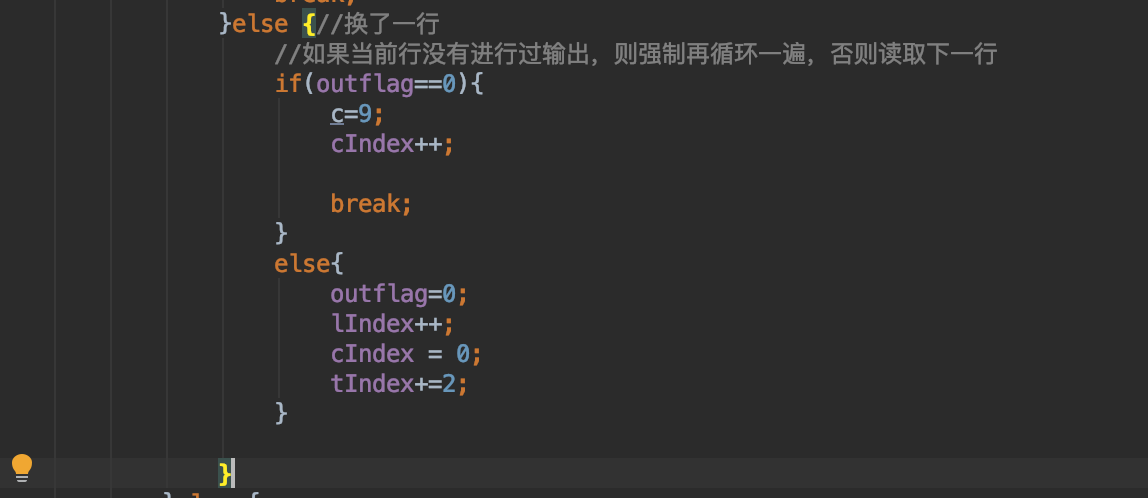
读到文件末尾输出EOF时tIndex-2；

1. 将所有关键字加入keywordSet中。
2. 新建calcuSet，并将运算符按第一个读取的符号分类，将读取完成后的lexme字符串与表calcuSet中的字符串对应并输出。
3. 在已有的DFA\_STATE\_ID\_0状态中添加对下划线的识别机制，切对于识别到单引号’、双引号”及常量前缀时进行相应的状态转换。
4. 新建判定函数
   1. isOct()用于判断是否为八进制字符；
   2. isDigitSymbol()用于判断是否为数字常量的前缀/表示八进制、十六进制所需的符号。
   3. isEscape()用于判断在已读取到\的前提下，当前字符是否与\构成简单转义符/八进制转义序列/十六进制转义序列
5. 由于在实验过程中，我将MyScanner输出与内置词法分析器输出对比后发现，当某一行只存在一个关键字（如else/goto），且后面没有其他结束符号（如；等），则扫描到下一行的\t字符时才会进行输出，导致索引号tIndex、行号lIndex、列号cIndex均错误。

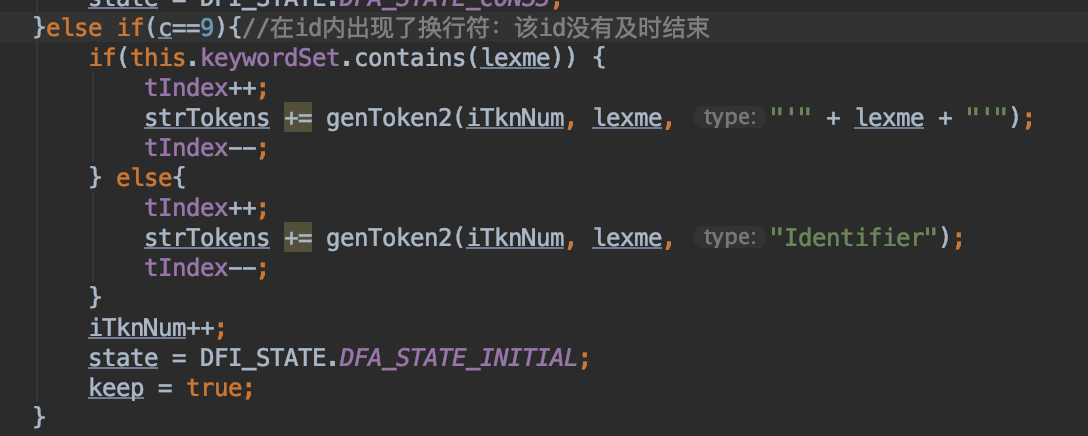




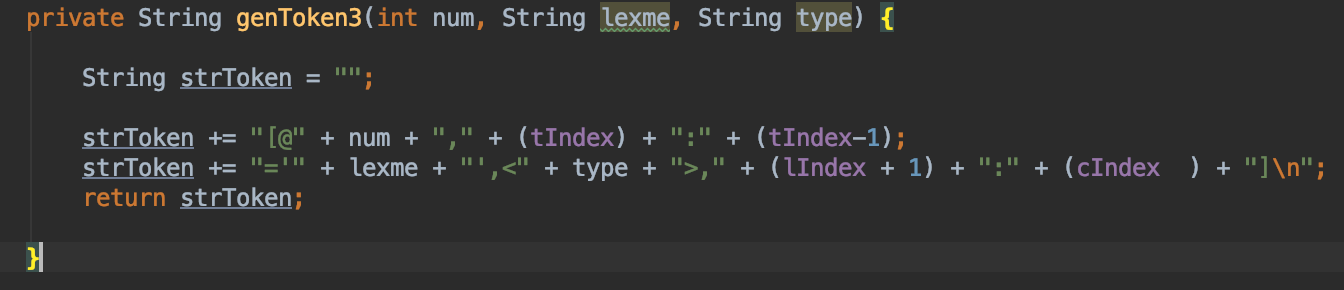
因此，我增加了一个输出标识符outflag用于表示该行是否以进行过输出，行数更新、列数清零时对outflag置零，在输出函数gentoken()中对其置1



如果当前行没有进行过输出，则在请求换行时将c换为制表符，强制再循环一遍（判断DFA\_STATE）

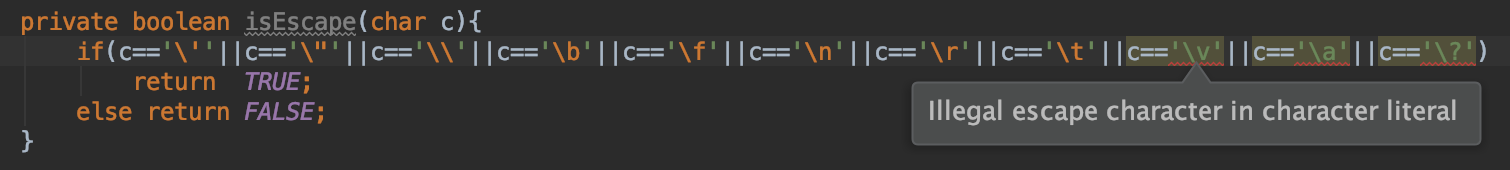


1. 由于结束符EOF的输出规则与其他token不同，为其编写新的输出函数gentoken3()

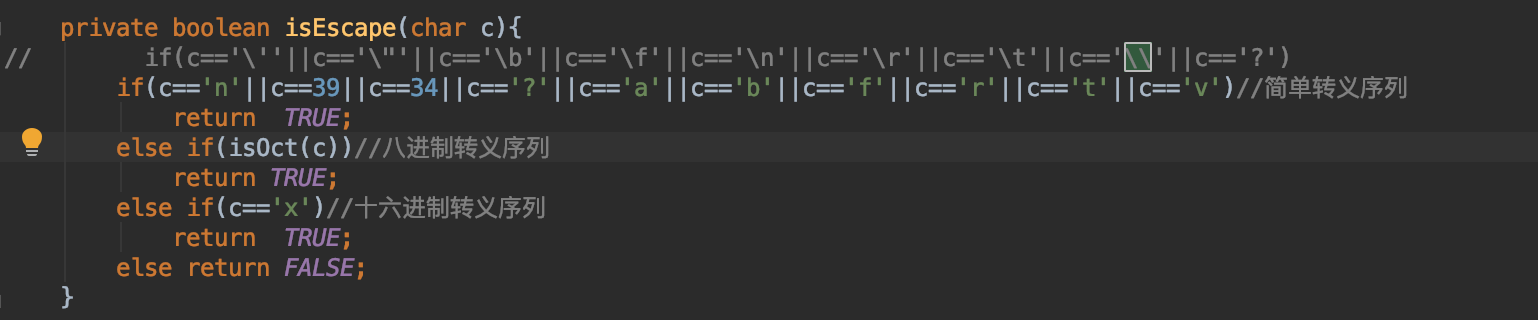


## 实验中遇到的问题及解决方案

1. 在对转义符进行识别时发现了两个问题，一是字符读取时并没有将转义符转义后存入当前字符c中，而是将\和后面的字符分别存入c中，所以按下图所示判断机制无法识别出简单转义符。二是\v,\a,\?被识别为非法字符。



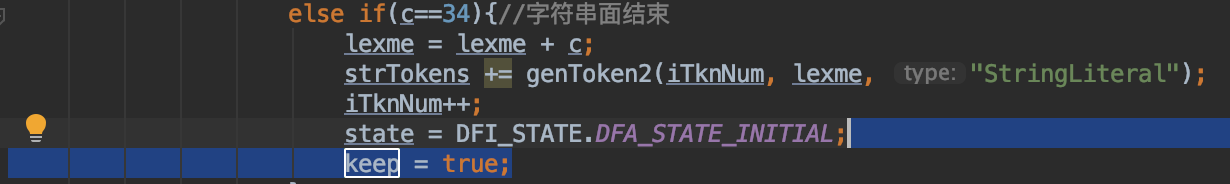
解决方法：依次识别\符号后进入判断是否为转义符状态，再识别其后面的字符，判断是否为转义符。



1. 对框架中原有的gentoken（）函数和gentoken2（）函数认识不够清晰，对于有些状态调用了错误的输出函数，导致输出索引不正确。

解决方案：gentoken（）函数适用于读到单个字符直接判断该token已结束，并进行输出的状态，如读到{,;,(等字符时。而gentoken2（）函数适用于读到了其他不合法字符从而判断当前token结束的情况。

如以下对于字符串面常量进行识别时进行了错误的调用，识别字符串面常量时是读到双引号”则判断该token结束，没有读到不属于该token的字符，应该调用gentoken（）函数进行输出。



1. 对于部分行没有结束标识时的错误已在上一小节中提及。

# 实验心得体会

本次词法分析器的构造主要基于BITMiniCC中框架，利用课堂上所学到的关于DFA构建词法分析器的知识，对示例词法分析器进行了改造和完善。

经过本次实验，我对c语言中各种常量、运算符、关键字等的定义有了更清楚的认识，加深了对于词法分析器的理解，也尝试了使用flex自动生成词法分析器（虽然最后没有采用这种办法），学会使用java语言直接编写词法分析器，对学到的知识进行了实践和体会。

即使已经尽力，面对测试源码也有了和内置词法分析器完全相同的输出，但程序仍有不够完善的地方，将持续进行改进。