词法分析器报告

学号：71Y16123 姓名：刘钊伟

# 一、目的

手工实现C词法分析器（C的子集）。

# 二、内容

处理C源程序，过滤无用符号，判断程序中单词是否合法，然后将单词与类别对应。

# 三、实现方法

1. 对源程序进行预处理（去除注释和换行符）
2. 将预处理后的程序分解为单词，这些单词可以分为五类：保留字、常数、标识符、逻辑运算符、界符。这个分解过程也可以理解为组合过程：将字符流组合成为单词。

相关文法如下：

标识符 🡪letter(letter | digit)\*

常数 🡪 (1-9)(0-9)\* | 0. (0-9) (0-9)\* | (1-9)(0-9)\* .？(0-9) (0-9)\*

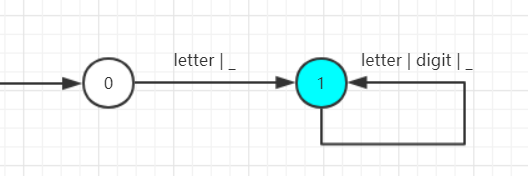
逻辑运算符 🡪 +、-、\*、/、>、<、=、>=、 <=、!=, ++, --, &, && 等

界符🡪 { } [ ] // /\* \*/ ( ) ， ；

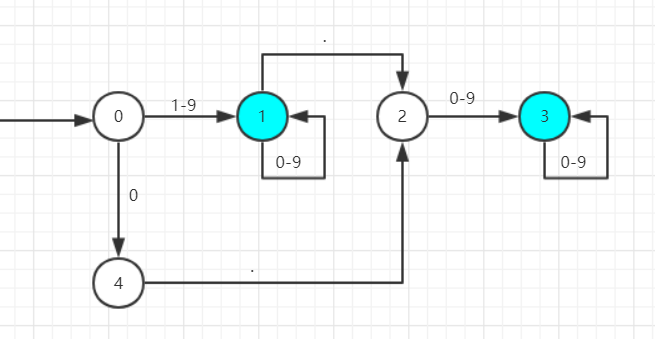
保留字：void, int, long, unsigned, short, float, double, if, else, do, while, for, break, continue, return

# 四、相关FA描述

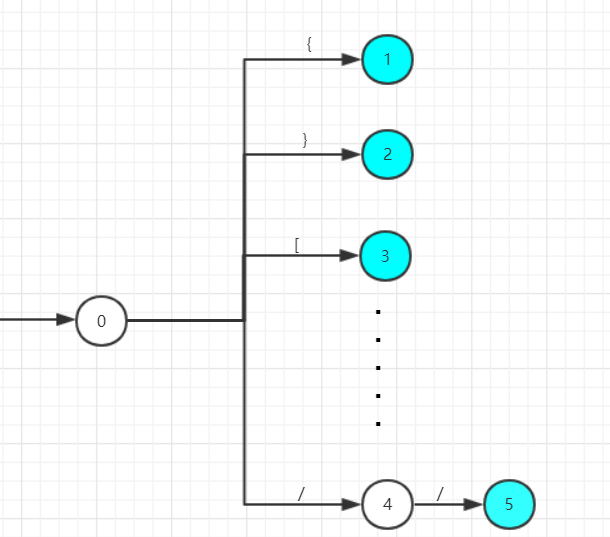
标识符：因为工具限制，使用底色为蓝色的状态作为终结状态



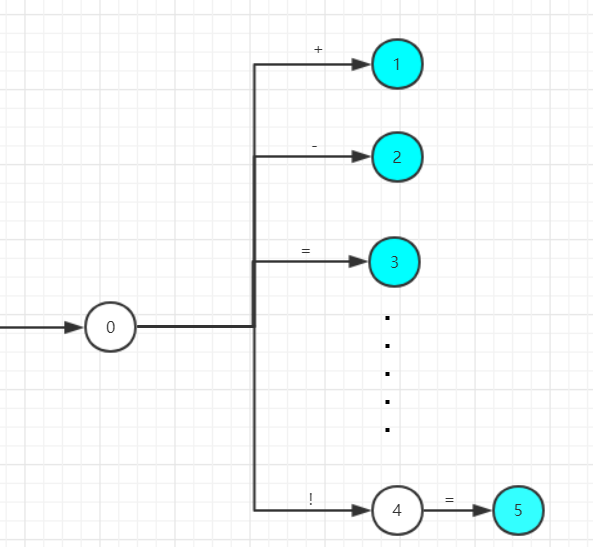
常数：



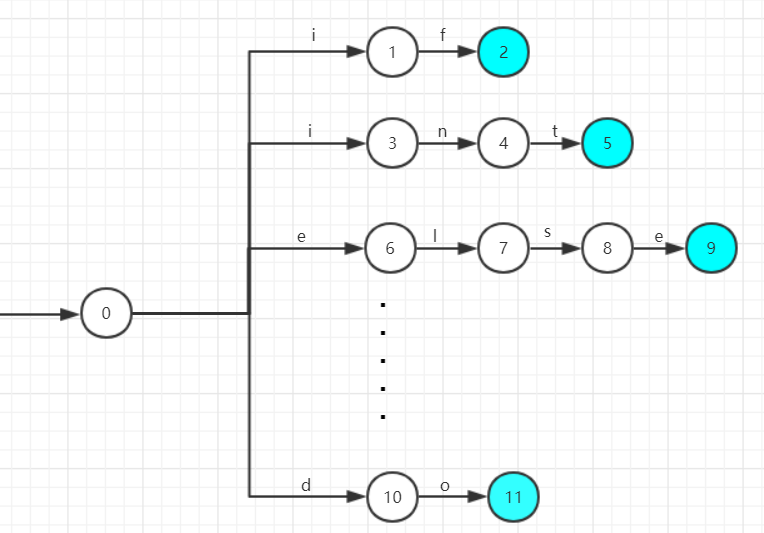
界符：（部分状态相似界符省略）



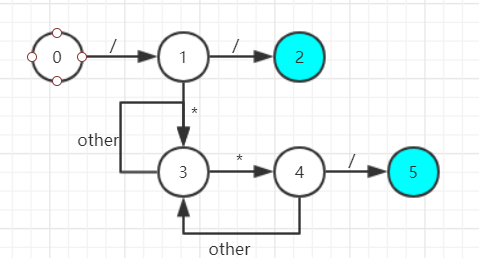
逻辑运算符：（部分状态逻辑运算符省略）



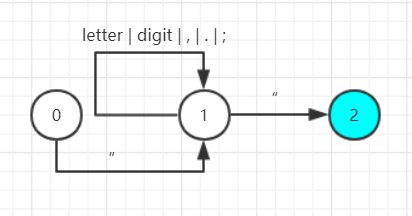
保留字：（部分状态相似保留字省略）



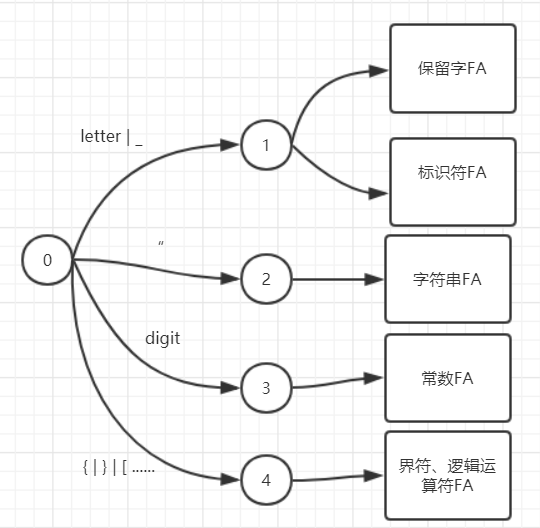
注释：



字符串：



整合FA：



# 五、核心算法描述

核心算法共分四部分：

1. 预处理部分（只去除注释）

FA见4-注释 图

读取代码🡪查找注释🡪如果是单行注释，就找到从注释符开始的下标和从注释符开始第一个换行符的下标🡪如果是多行注释，就从找到开始注释符的下标和结束注释符的下标🡪通过上面获得的下标，找到注释部分🡪通过replace函数将该部分替换为空

1. 标识符与保留字部分
2. 字符串部分
3. 数字部分
4. 界符与逻辑运算符部分

上面四部分都在主要词法处理函数中，因此一起写到下面：

每次分析器读取一行代码，对该行代码进行识别与错误分析

FA见 第四部分

整体思想是通过单词的开始符号来区分单词的大概类型，然后通过不同类型的FA来识别单词的具体类型；

因为标识符与保留字的模式很相似，所以这两个被分到了一个大类里面，先找出单词，再查看该单词是否在保留字列表中，如果在，则将该单词存储到token table中，如果不在，就将单词进行标识符的匹配，如果还无法匹配，则将报错。

字符串部分以 ‘ “ ’ 开头， 以 ‘ “ ’结尾，通过字符串的FA来匹配字符串，如果没有封闭，则字符串会报错

数字部分、逻辑运算符部分和界符部分都是通过类似与字符串的匹配模式，来匹配单词。

# 六、运行情况

对于类c语言代码

int main()

{

/\* aaaaa \*/

"aaaaa"

int id;

int \_id;

float p = 1.23;

char c = 'a';

if p == 1.23

{

printf("hello");

}

else

{

print("hhhh")

}

while (c == 'a')

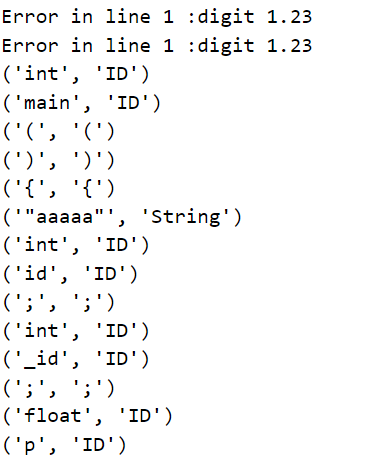
{

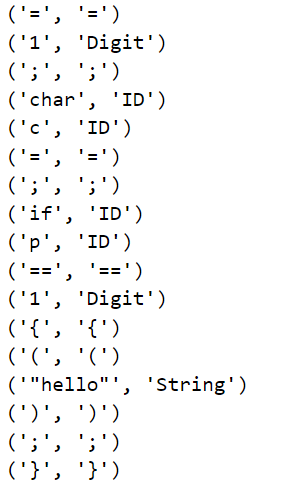
printf('a');

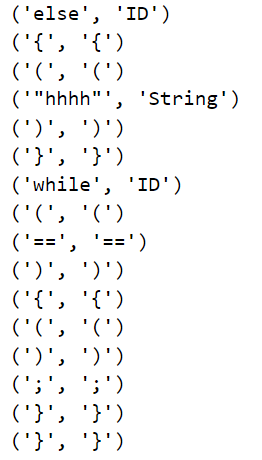
}

}

词法分析器的使用情况如下







# 七、出现的问题

1刚开始对于token如何存储和状态的表示都使用参数传递的方式，但是因为参数过多以及作用域复杂的问题，出现很多错误，后来使用全局变量来作为参数，状态在每次scan的时候置零，成功解决了这个问题。

2错误分析程序没有实现预想中的效果，对于没有封闭完全的字符串，无法识别出错误；

3对于错误的行数，一直显示为 错误在第一行，不知道如何处理

# 八、感想

词法分析器是编译器的基础，它面向的对象是单个字符组成的字符串流，它是语法分析器的前置，通过本次实验，我对词法分析器的原理有了更加深入的了解，对从正则表达式，到NFA，到DFA到最小化DFA这个流程更加熟悉，这次实验实现的词法分析器在识别单词方面较为成功，但是在错误处理方面有些问题，需要改进。