编译原理实验一

词法分析扫描器设计实现

姓名	马浩宇	学号	19335154

一、实验目标

手动设计一个词法分析器。首先需要了解所选择编程语言单词符号及其种别值，要求实现的功能为：

输入一个C语言源程序文件demo.c

输出一个文件tokens.txt，该文件包括每一个单词及其种类枚举值，每行一个单词。

二、设计过程

首先需要在针对具体的程序文件进行词法分析之前，需要根据C语言的语法预先定义好关键字和界符，在此次实验中，预先定义好的关键字及其对应的种别码如下表所示：

auto	**04**	**void**	**05**
char	06	const	07
double	08	float	09
int	10	long	11
if	12	else	13
for	14	while	15
continue	16	break	17
switch	18	case	20
main	21		

同时该表也为此次实验设计的词法分析器所包含可以识别的关键字的范围。接下来是关于界符表pt的定义如下表所示，对应的即表示此次实验中支持识别的界符范围：

(	30	)	31	[	32	]	33
{	34	}	35	+	36	++	37
+=	38	-	39	--	40	-=	41
*	42	*=	43	/	44	/=	45
%	46	<	47	>	48	=	49
<=	50	==	51	>=	52	&	53
|	54	^	55	&&	56	||	57
//	58	/*	59	*/	60	,	61
;	62	<<	63	>>	64	->	65

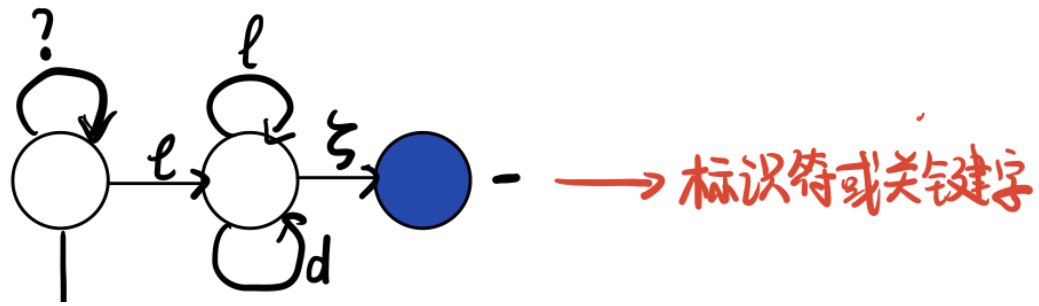
在词法分析器的实现中，通过`map<string,int>`的形式分别对上面的两个表进行表示和储存。同时将标识符的种别码定义为00，将字符常量定义为01，字符串定义为02，数字定义为03。

自动机的设计

在提前设定好了关键字表和界符表之后，接下来设计自动机对程序中的词进行识别分析，本次实验中打算识别的单词范围主要包括：**普通标识符、设定好的关键字及界符、数字常量（整数、小数、科学计数法）、行注释及块注释、字符串和字符常量的识别。**接下来将分别介绍对应识别内容自动机的设计以及代码实现思路（下面分析过程图中的状态标号可能与实现过程中有所不同）：

标识符及关键字

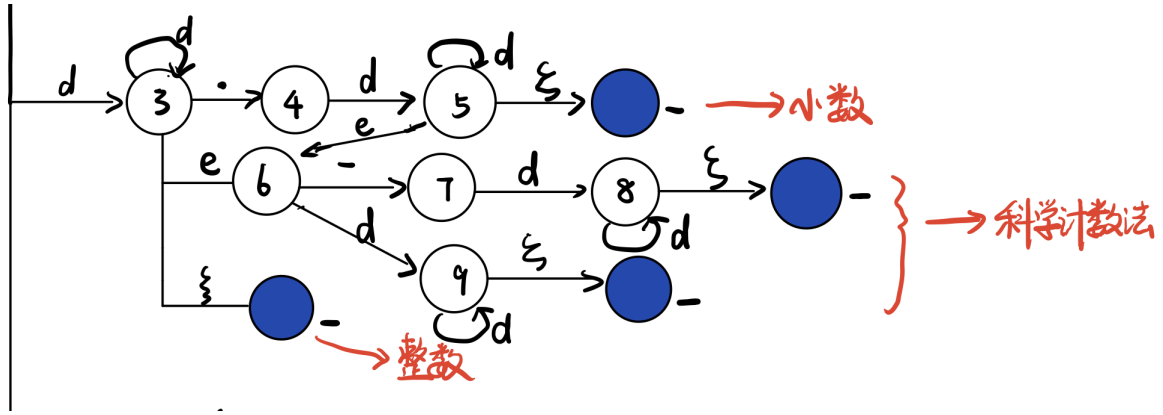
由C语言设计标准中标识符只能由下划线、字母和数字构成，并且标识符不能和已经设定好的关键字重复，所以在设计词法分析器过程中可以将关键字看作为一种特殊的标识符，两者的自动机表示相同，其对应的自动机设计如下图所示：



其中对应的？为空格、换行符等需要跳过的词，l表示字母，d表示数字，ζ表示除了表示出的字符外其他的字符，该自动机的含义为当处理字符串过程中如果读取到字母或者下划线则进入下一个状态，并且可以重复读取字母、下划线或者数字，当读到除这些字符以外的字符时，则进入终止态。当获取了这样一个字符序列后，首先默认其为标识符，接下来通过`map.count()`函数来判断该字符串是否在关键词表中出现过即是否是关键词，如果是关键词则通过map对应的键值对找到其对应的种别码进行储存，如果不是则通过`find()`函数查看该标识符是否出现在iT表中，如果是首次出现，则还需要将其储存进入对应的iT表中。

数字常量

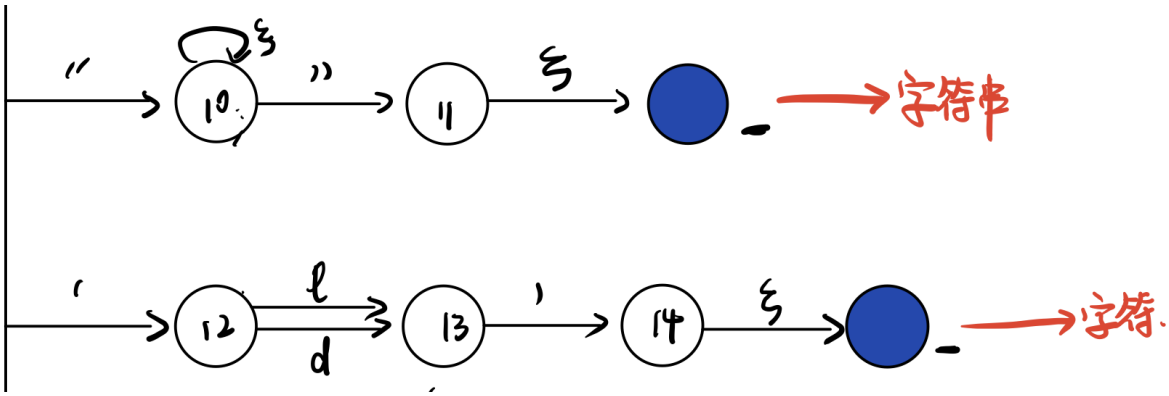
可以识别出的数字常量包括整数、小数和科学计数法表示，对应的自动机设计如下所示：



对应首先从初始状态1读取到数字则进入状态2可以重复读取数字，如果此时读取到不是小数点、字符e、数字，则表示该数字读取结束为一个整数，则进入终结态。接下来如果读取到小数点则进入状态4，在该转态如果继续读取到数字则表示该数字常量表示一个小数，如果读取到的不是一个数字的话，则该格式不符合C语言的要求，进入报错状态，进行报错提示。同样在状态5可以反复读取数字，接下来如果读取到字符e，则进入状态6，表示为一个科学计数法，如果是其他字符则进入终止态。同样在状态3读取到字符e也可以进入状态6同样表示科学计数法，在e后面可以紧跟着一个负号，接下来即重复的读取数字，直到识别到的字符不为数字表示该数字常量读取结束。读取到的数字常量，同样除了按照对应的类别码储存到tokens结果数组中，还需要储存到对应的CT表中。

字符串及字符常量

可以识别到“ ”，' '对应的字符串（包括空串）和字符，其对应的自动机设计如下图所示：



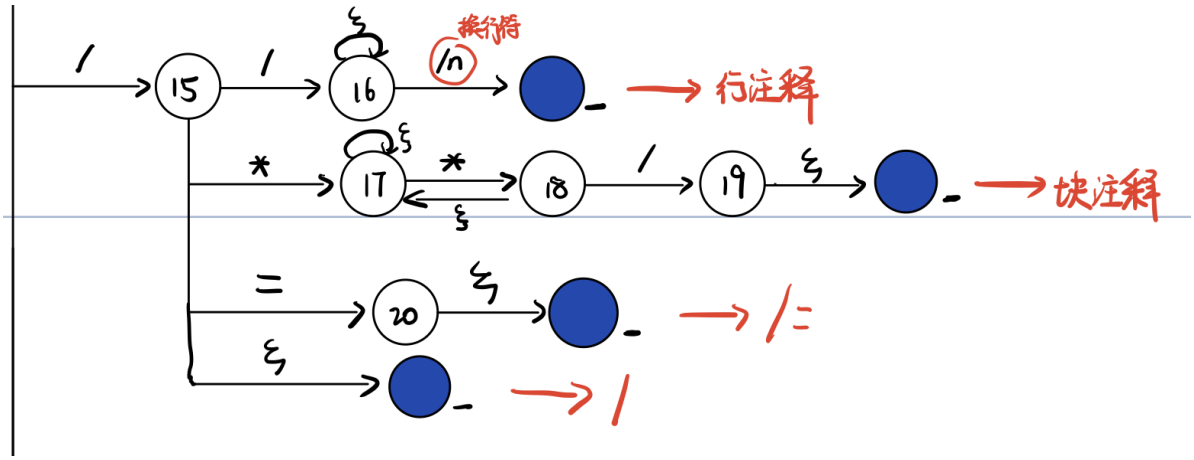
其中设计和ppt上略有不同，ppt上的字符串识别自动机无法识别到空串。所以在进行字符串的识别进行了一些修改，对应的从初始状态读取到`"`或者`'`时分别进入状态10和12，值得注意的是字符串可以使任意长度的，而字符常量只能由一个字符组成，否则则会产生报错提示。分别在读取到下一个双引号或者单引号时准备进入终止状态。

在设计读取字符串时，比如要考虑到转义字符存在的情况，所以当在读取字符串和字符时如果识别到`\`转义字符时，需要自动将下一个字符不能将其当做具有状态跳转意义双引号或者单引号进行读取。进而实现了可以识别转义字符的功能。

与上面相似的，在完成对应的字符串和字符读取后，除了按照对应的种别码储存在tokens结果数组中之外，还需要将其储存在sT和cT数组中。

行注释及块注释

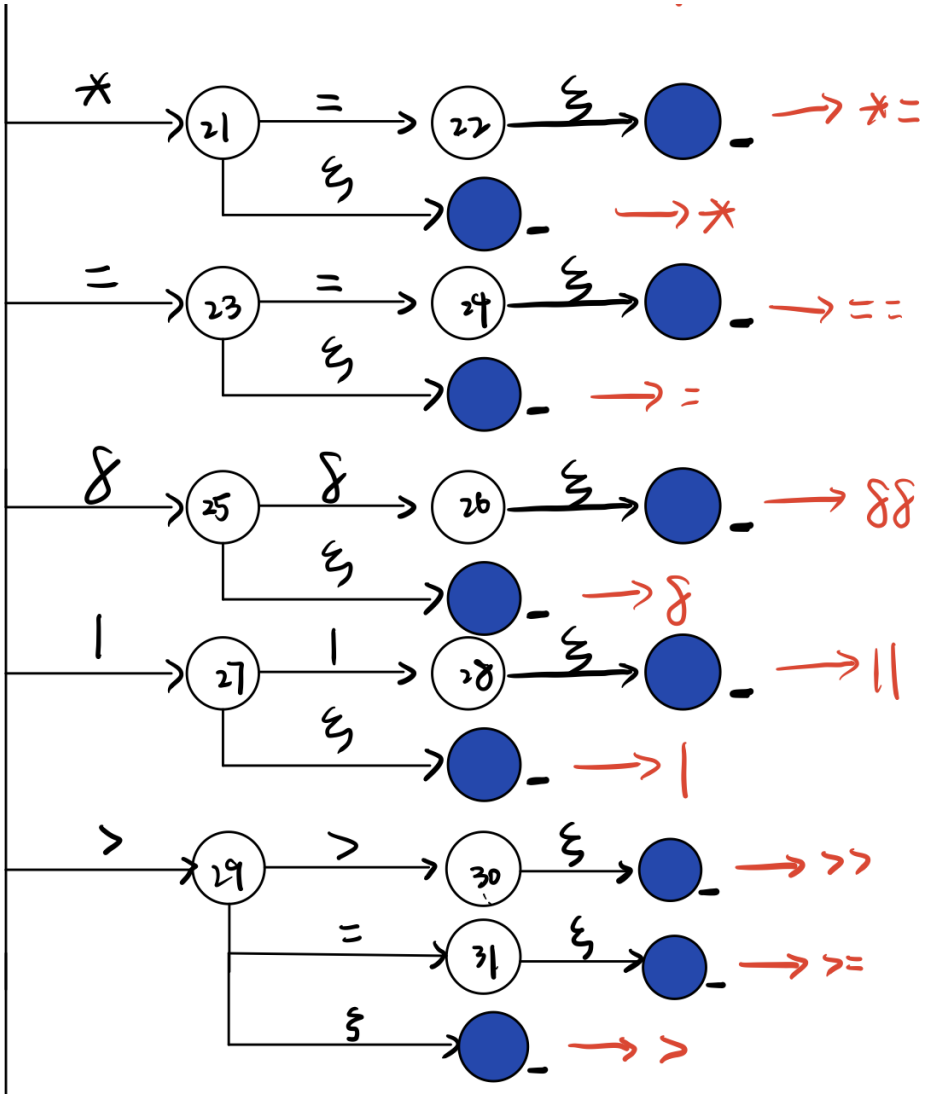
在本次实验设计标准中，注释中的内容不会被当做标识符、关键字或界符等进行记录，这就要求当进入了读取注释的状态时，要忽略后面的字符内容，直到从对应的注释状态中退出后，才能重新进行记录，具体关于注释的自动机如下所示：



从上面自动机可以看出，从初始态如果连续读到字符//或者/*时分别进入行注释状态16和块注释状态17，在注释状态时，无论读到任何字符都不进行记录，直到读到换行符即退出行注释状态，而连续读到*/时字符时，结束块注释阶段，在对应的终止态时分别将对应的行注释符和块注释符和对应种别码存入tokens。

一般界符的识别

一般界符是指基本运算符，识别的大体思路类似，基本上为按流程识别到对应的单目运算符后，在根据后续的字符判断是否为双目运算符，并对应不同的终止态，但是在通过代码实现时，发现可以各个运算符共享识别到单词后缀符进入终止态的状态，因为在原本的设计思路中需要通过不同的终止态来判断对应的运算符分别是什么，而在实现过程中通过map储存所有界符的方式，只需要在终止态通过对应界符的储存的字符串即可得到对应的种别码，并储存到tokens中。下面以几个基本的运算符分析的自动机为例：



代码设计思路

本次实验通过C++实现对应的词法分析器，将对应的词法分析器封装成唯一Lexer类。其中对应的函数和变量基本如下所示：

```
class Lexer {
public:
    //进行词法分析的函数,返回tokens
    vector<pair<string,int>> word_analysis(string code, int& error_flag);
    //分别输出对应的表
    void show_iT();
    void show_CT();
    void show_sT();
    void show_cT();
private:
    //判断字符是否为数字
    bool isDigit(char c);
    //判断字符是否为字母或下划线
    bool isLetter(char c);
    //自动机状态控制转换函数
    void state_change(int& state, char ch, int& error_flag);

    map<string, int> kt;
    map<string, int> pt;

    vector<string> sT;
```

```
    vector<string> cT;
    vector<string> iT;
    vector<string> CT;
};
```

其中最重要的函数为`word_analysis()`，其主要思路如下所示：

```
vector<pair<string, int>> word_analysis(string code, int& error_flag) {
    //初始化变量和结果数组等等
    ...
    //开始遍历识别代码
    for(int i = 0 ; i < code.size() ; i ++) {
        //记录状态变化
        state_before = state;
        state_change(state, ch, error_flag);//根据当前字符变换状态
        //终止态为state=0的时候
        if(state){
            ...//非终态和非注释态根据需要记录字符
        }
        else{
            ...//根据state_before判断对应种别码的类别
            write_in_tokens();//存入结果数组
            reset();//重新初始化对应的临时字符串和状态重新设置为初始态
        }
        if(error_flag) ...//如果错误码为1表示出错，需要结束函数并报错
    }
    //返回tokens结果数组
    ...
}
```

状态转移函数`state_change()`则是根据上面绘制的自动机通过if else语句进行状态切换的函数，表示用于切换当前对应的状态，通过一个整数state来进行标识不同的状态。同时通过不同的state标号，可以在获取字符的时候判断当前属于什么状态，比如说如果识别到当前状态是读取注释的状态的话就不再记录当前读到的字符。在state=0的终止态时，可以通过state_before变量的值来判断识别到的该词为什么种别，从而分种类对其进行后续的处理。在具体实现对应的状态时，发现因为已经提前储存好界符表的缘故，所以不需要设计很多的终止状态，从而大大减少了自动机状态的数量，减少了重复的代码量。同时还通过添加error_flag变量来防止错误的出现，比如如果只有/*符号却没有注释块结束的*/符号时便会报错，同时在识别到如123.123.123这种错误的变量时也会清空tokens，并进行报错提示。

上述即为此次词法分析器实验的自动机设计和代码设计原理，整个词法分析器完整的自动机设计在报告的最后进行展示。

三、结果展示

结合ppt提供的`demo.c`的样例并结合设计添加的新功能稍加修改得到的测试`demo.c`如下所示：

```
int main() {
    //sdsadasd
    a += 1;
    int a = 1, d = 2.4e-19, c;
    if(a <= d) {
        c = a;
        a = d;
        d = c;
    }
```

```
    char ch[10] = "ok\\ds";
    char x, y = 'a';
    c = a + d;
}
/*sdada*/
```

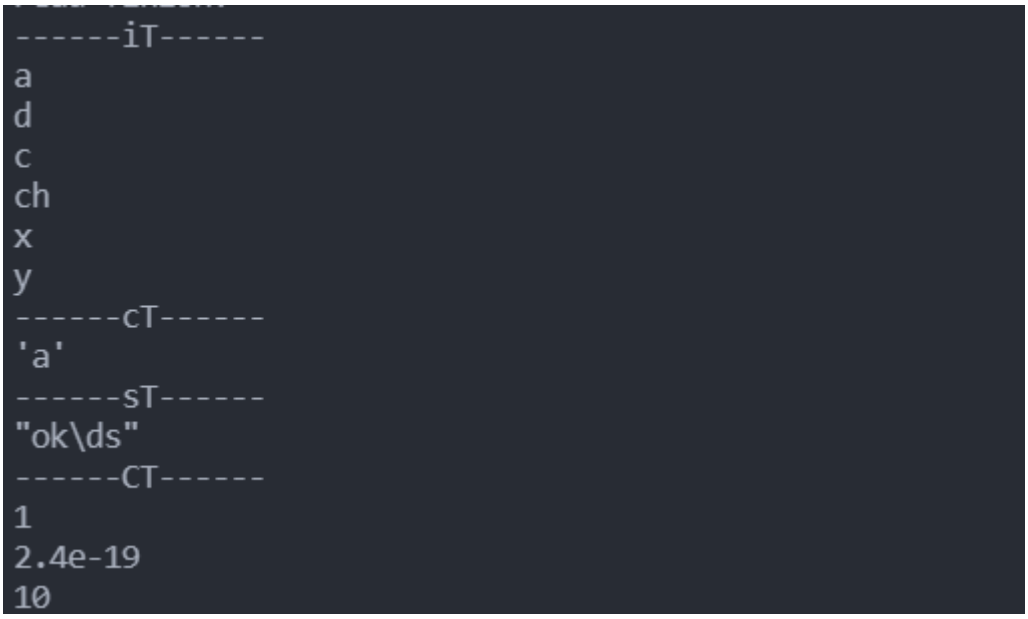
其中包含了对添加的科学计数法的测试，以及行注释和块注释的测试，还有就是对字符串中出现转义字符时的处理，结果呈现在`tokens.txt`中，运行的到的tokens部分截图如下所示：

<int ,10>
<main ,20>
<(,30>
<) ,31>
<{ ,34>
<// ,58>
<a ,0>
<+= ,38>
<1 ,3>
<; ,62>
<int ,10>
<a ,0>
<= ,49>
<1 ,3>
<, ,61>
<d ,0>
<= ,49>
<2.4e-19 ,3>

可以看到成功识别到了行注释符，并没有将后续对应的注释识别为标识符，同时可以正确识别设定好的关键字和界符还有标识符等等，也可以看出在最有一行的科学计数法也成功识别成了数字。

<"ok\ds" ,2>
<; ,62>
<char ,6>
<x ,0>
<, ,61>
<y ,0>
<= ,49>
<'a' ,1>
<; ,62>
<c ,0>
<= ,49>
<a ,0>
<+ ,36>
<d ,0>
<; ,62>
<} ,35>
</* ,59>
<*/ ,60>

从这部分结果可以看出此时第一行可以正确处理字符串常量中的转义字符，同时可以看出此时在最后的两个块注释符也被成功识别。



同时可以看到此时对表中的内容进行展示也都是正确的结果，词法分析器的基本功能得到完成。

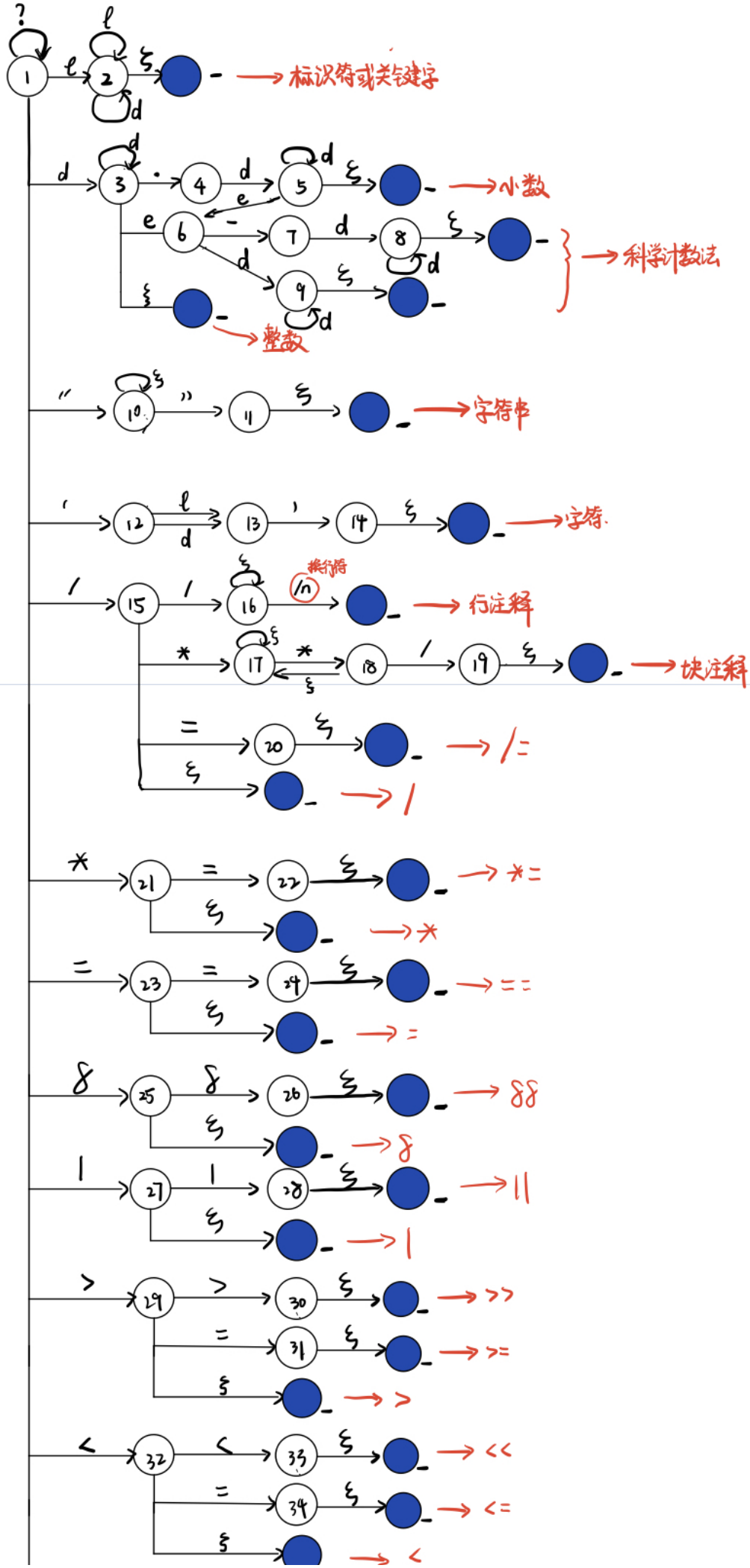
四、心得及总结

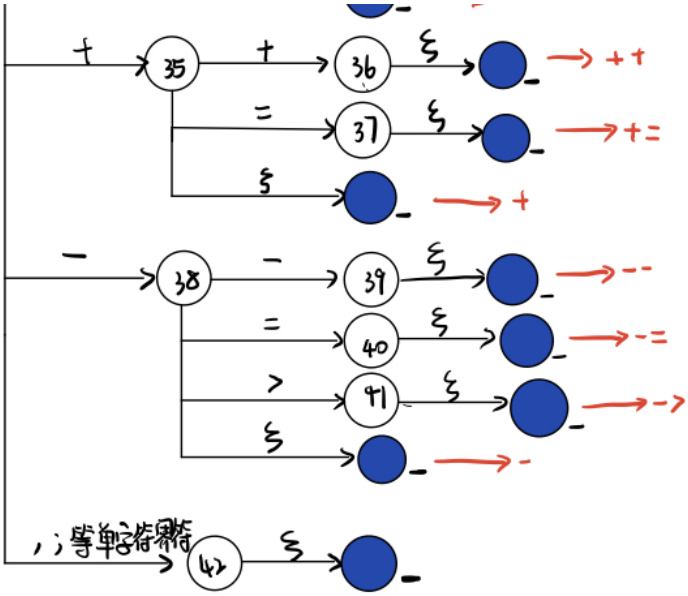
本次词法分析器的设计主要难点在于前期对功能的设计和自动机的绘制，在进行实现时的难度并不大。通过本次实验了解了C语言的语言特点，通过实践熟悉了对自动机的设计，以及自动机在处理任务时的应用场景。动手实践与课上的内容相结合，对词法分析的内容有了更好的理解，同时虽然设计了很多功能，但是在后续语法分析的过程中可能也会有所删减或者增加。

首先就是对于报错的内容现在无法区分具体的错误，现在只可以识别出不符合规定的数字常量，或者结束时缺少不位于终止态的问题，希望在后续的实验中能够完善报错机制，比如说可以针对具体的错误输出对应的语句进行提示。

本次实验的完整代码见https://github.com/WsgDcb/compiler_lab/tree/master/lab。

本次实现完整的自动机设计图如下所示：


标识符或关键字
小数
科学计数法
整数
字符串
字符
行注释
块注释
/=
/
*=
*
==
=
&&
&
||
|
>>
>=
>
<<
<=
<


+
35
+
36
=
37
+
+
+=
+
-
38
-
39
=
40
>
41
--
-=
->
-
;
42