编号：(2021-2022-2)-3150520011025-9、10

实习	一	二	三	四	五	六	七	八	九	十	总评	教师签名
成绩												

武汉大学计算机学院

《编译原理》课程

词法分析

实习报告

编　　号：CSA20

实习题目：构建一个小语言的词法分析程序

专业（班）：计科 10

学生学号：2019302060078

学生姓名：梁艺馨

任课教师：杜　卓　敏

目录

词法分析

1.语言形式化描述

实现 **c 语言**的词法分析，将 c 语言的相关词语符号分为以下六类

分类

界符：{} [] () ； "" ''等

运算符：+ - * / += -- *= -= == & && | ||,主要包含的运算符类别有算术运算符，关系运算符，逻辑运算符，位操作运算符，赋值运算符

数字：包含了二进制数字，八进制数字，十进制数字，十六进制数字，小数，科学计数法

标识符：c 语言规定，标识符只能由字母 A-Z，a-z，数字 0-9 和下划线_组成，并且第一个字符必须是字母或下划线，不能是数字

关键字：C 语言本身定义的常用关键字集合

注释：C 语言支持单行注释和多行注释：

单行注释以//开头，直到本行末尾

多行注释以/*开头，*/结尾，可以一行或者多行

2.单词编码表

iT 标识符	00	cT 字符	01	sT 字符串	02
cT 常数	**03**				
KT 关键字					
auto	04	short	05	int	06
long	07	float	08	double	09
char	10	struct	11	union	12
enum	13	typedef	14	const	15
unsigned	16	signed	17	extern	18
register	19	static	20	volatile	21
void	22	if	23	else	24
switch	25	case	26	for	27
do	28	while	29	default	30
sizeof	34	return	35	mian	36
PT 界符					
+	37	-	38	*	39
/	40	%	41	++	42
--	43	>	44	<	45
==	46	&&	50	||	51
!	52	&	53	|	54
~	55	^	56	<<	57
>>	58	=	59	+=	60
-=	61	*=	62	/=	63
%=	64	&=	65	|=	66
^=	67	>>=	68	<<=	69
(	70	)	71	[	72
]	73	{	74	}	75
'	76	"	77	;	78
,	79	!	80	//	81
/*	82	*/	83	?	84
.	100				

3.状态转换图

初始字符读取状态为 state==1

将单引号和双引号中的字符单独取出，放置在对应的 cT 字符与 sT 字符的容器中，所以归属于界符的状态主要有三种：

state==2 普通界符

state==3 读取到的界符为单引号，需进行进一步处理

state==4 读取到的界符为双引号，需进一步处理

运算符：state==5，此时运算继续向下读取

数字：由于数字的读取中，加入了一些比较复杂的进制识别，因此数字也需要分为多个状态进行处理

state==6 当前字符识别一直为整数（包括相关进制代表符号 0b 或 0x 等）的时候 state 为 6

state==8 当读取字符的过程中出现小数点，此时将状态转换为 8，便于后续解码处理小数

state==9 当读取字符中出现科学计数符 e|E 时候，此时状态转换为 9，便于后续解码处理科学计数

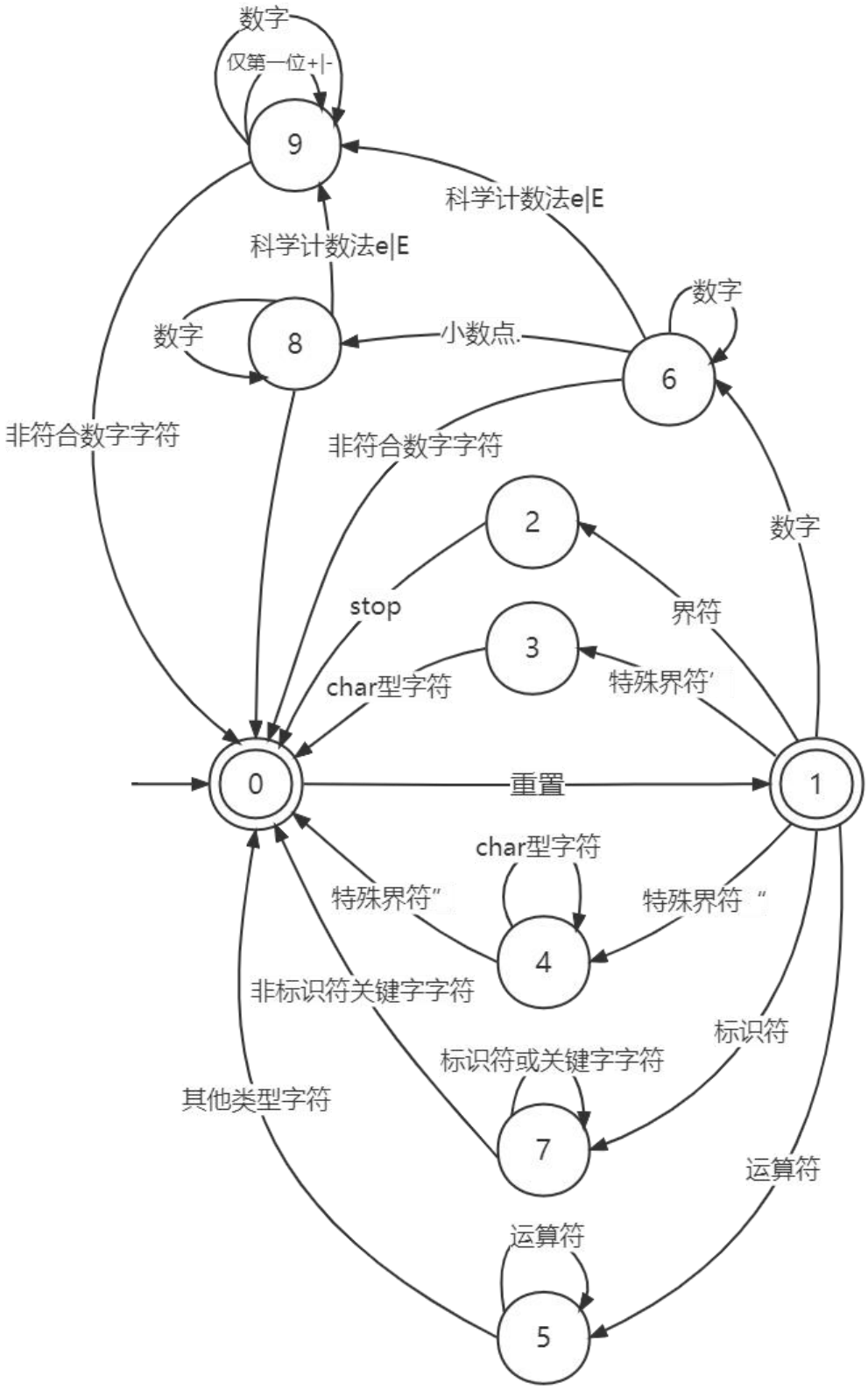
标识符，关键字，state==7：由于构建条件较为相似，我们置于一起进行读取，在后续解码过程中再进一步划分

state==0 当前状态需要结束的时候，此时所有状态归为 0，完成当前单词的读取，并进入解码模块。具体结束情况举例如下：

当前状态转换到另一种大类状态中，如界符后续读取到数字字符，则当前界符读取结束

字符读取到空格等无关字符，则当前字符读取结束

根据上述状态划分构建自动机 DFA 如下：


数字
仅第一位+|-
9
科学计数法e|E
科学计数法e|E
数字
8
小数点.
6
数字
非符合数字字符
非符合数字字符
2
数字
stop
界符
3
char型字符
特殊界符'
0
重置
1
char型字符
特殊界符"
特殊界符“
4
非标识符关键字字符
标识符
标识符或关键字字符
其他类型字符
7
运算符
运算符
5

4.词法分析器算法

核心函数与参数介绍

```
//参数
vector<char> cT;    //保留字符
vector<string> sT;  //保存字符串
map<string , int> PT;   //PT 符号
map<string , int> KT;   //KT 关键字
vector< pair<string , int> > res;   //保存最后的读取结果

void init_all_map(); //初始化映射表
int state_change(int state , char ch); //状态转化
int get_state_ch(char ch);  //获得当前字符的状态
int State2Code(int state_before);   //状态转编码
int parse(int code , string token);//解析编码
void show();    //展示获得的输出内容
bool check_legal_num(string token);//检查是否为合法的数字
bool all_zero(string token); //检查输入全为0 的时候可能出现的错误
void Error_print(int state , int line);//输出错误信息
```

函数介绍：

初始化映射表：init_*all*_map()

建立两个 map（unorderd_map）,完成上述枚举值表的映射

状态转化函数：state_change(int state , char ch)

参数介绍：state:当前所处状态，ch 当前字符 ， 返回值 int，返回当前所应返回的状态

在这个函数中我们需要按照前面展示的状态转化 DFA 来完成对于函数的构建

状态转化介绍：

1->other：当前状态为 1，默认为此时该字符前面所有字符已经完成状态转化，此时直接根据当前的 ch 字符，调用 **get_*state*_ch(char ch)**，获得当前字符的状态

2->0：观察状态转化表，当前状态为 2 的时候说明此时为普通界符，由于我们界符表中的所有字符均为单个字符，因此当获得该字符时，直接返回 0

3 状态转化：这里加了一个 flag_cT 标志进行数字/字符处理，当第一次遇到 cT 时会将 flag_cT 置为 1，第二次会将 flag_cT 置为 2。在这里，如果 flag_cT 是 2，那么就将 flag_cT 置为 0，返回 0，否则就是第一次遇到，将 flag_cT 置为 1，返回 0

状态 4 同状态 3，也加了一个 flag_sT 进行字符串处理

5 状态转化：对运算符进行处理，遇到其他类型字符时返回 0

6 状态转化：

1.6 的状态转化有多种可能，读取到字符为 e 的时候，此时为数字状态读取到科学计数法，状态转为 9，进行科学计数

2.读取 ch 为数字，此时返回 6 继续读取

3.读取非符合状况字符，此时返回 0，结束数字读取

7 状态转化：对标识符或关键字进行处理，当遇到非标识符或关键字字符时返回 0

8 状态转化：如果是数字，返回 8，如果是 e|E，返回 9

9 状态转化：第一个字符是进行符号处理，当第一个数字不是符号时，返回错误-7，后续是数字处理，当遇到非数字字符时，返回 0

获取当前字符对应的 state 值：get_*state*_ch(char ch)

将字符进行简单分类，思路如下

判断能否识别

判断字符是否为空或者\n，如果是，结束读取。

检查是否为 PT 字符，如果编码为 76（’），77（”），100（.），转换到对应状态 3，4，8，如果字符在 0-9 之间，转到状态 6，其余情况转到 7 进行标识符或关键字处理

状态转编码 State2Code(int state_before)

根据当前状态转为对应 code 编码即可，重点是下面的解析编码

解析编码：parse(int code , string token)

根据 code 值完成解码，记录在 token 中。我在这里完成了对于注释部分的处理，借助全局函数识别 token 保存到的字符为注释符号/*，根据判别完成对相关注释的处理，而对//字符进行判断的时候则放在函数识别界符中完成处理，只需忽略行后续字符即可

其他编码处理：处理界符只需直接调用映射表，而处理数字时，由于涉及到进制，因此我建立了几个函数用来检查数字的正确性：

```
bool check_legal_num(string token);//检查是否为合法的数字
int dif_base(char first , char second); // 根据前两位计数，获得此时的数字进制
bool all_zero(string token); //检查输入全为0 的时候可能出现的错误
```

核心逻辑：此时 case 到 code 的编码为 5，按照数字解析，调用函数 check*legal*num()检查是否为合法数字：当为合法的时候，则正常存入数字，若不合法，则返回此时的错误编码，结束读取。

检查思路：

先检查数字字符串的 size ：当数字长度小于 2 时，由于进入时已经确保第一位数为 1，则一定正确

检查数字字符串的前两位，返回将当前数字字符所用进制

根据不同的进制，确定数字所能取值的范围，并完成此时数字检查

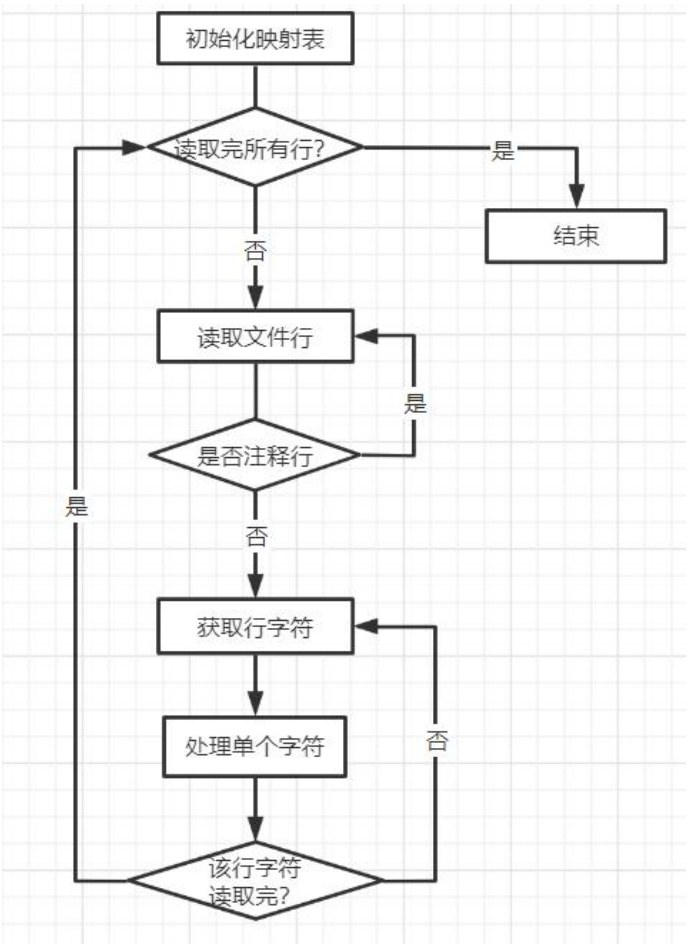
错误定义函数: Error_print(int state , int line)

一共定义了如下七种错误，对错误状态和错误行进行输出：

```
-1:"Error : Your input contains illegal characters in Line : "
-2:"Error : Contains multiple characters in a variable of type \"char\"
in Line : "
-3"Error : Contains unparsed characters  in Line : "
-4:"Error : The representation of numbers is wrong in Line : "
-5:"Error : Incorrect use of related symbols in Line : "
-6:"Error : Incorrect decimal representation is used in the statement in
 Line : "
-7:"Error : Incorrect use of scientific notation in Line : "
```

代码运行逻辑：

流程图：



用伪代码进行描述：

```
//line：所有行
//line.size:总行数
//sline:每一行的字符串
//flag_anno:单行注释//


    init_all_map();//初始化符合关键字的映射表
    for i=0 to line.size{//每次读取文件一行，便于进行处理
        sline<-linep[i]
        for j=0 to sline.size{
             if flag_anno=1{ //检测,检测是不是注释行
                state = 1
                token.clear()
                flag_anno = 0
                break
            }
            ch <- sline[j]//获取当前字符
```

```
            state_before <- state
            state <- state_change(state,ch)//状态转化

            if state > 0{//当状态不为 0 时，将当前字符加入 token 中
                token += ch
            }
            elif state = 0 {
                state <- 1//状态为 0 时，按照状态转化表将状态置为 1
                code <-  State2Code(state_before)//状态转编码
                check_parse <- parse(code, token)//根据 code 值完成解码
                if check_parse < 0 {//报错处理，输出错误信息和报错行
                    Error_print(check_parse,i)
                    return 0
                }
                state <- state_change(state,ch)
                token.clear()//state 为 0 的时候将 token 清 0，对下一个词语符
号进行记录
                if state {//状态不为 0 时继续进行下一次处理
                    token += ch
                }
                else{
                    state = 1
                }
            }
            if state < 0 {
                Error_print(state,i)
                return 0
            }
        }
    }
}
```

5.测试计划

正确用例以下面的源程序进行说明

标识符和关键字在下方有展示，如 a，asd_a12，int，char 等

```
int main()测试普通界符()
{
int a  = 132.1321;测试小数
a += 00;
a += 00101;以上两条测试八进制，特殊：00，000 会报错
a = 0x10a1;测试 16 进制
a = 10.0e+10;
a = 10e-123;
a = 10e+123;以上三条测试科学计数法
a = 123456;
```

```
long b == 0x8;
double c >>= 0b101;测试位运算符，二进制
int d = a + b;测试算术运算符
char q = 'a';测试单引号界符
string asd_a12 = "asdads";测试双引号界符+关键字
scanf("%d " , &a);
printf("%d" , a); //test 测试单行注释
/*aaaaaaaa
aaaaa //test
test
*/
测试多行注释

return 0;
}
```

运行结果如下：

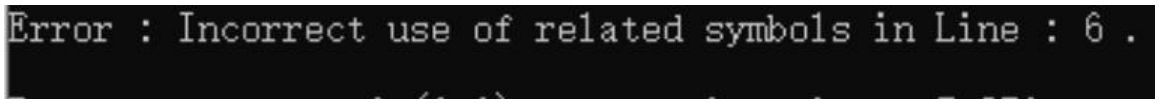
```
< int , 6 >
< main , 36 >
< ( , 70 >
< ) , 71 >
< { , 74 >
< int , 6 >
< a , 0 >
< = , 59 >
< 132.1321 , 3 >
< ; , 78 >
< a , 0 >
< += , 60 >
< 00 , 3 >
< ; , 78 >
< a , 0 >
< += , 60 >
< 00101 , 3 >
< ; , 78 >
< a , 0 >
< = , 59 >
< 0x10a1 , 3 >
< ; , 78 >
< a , 0 >
< = , 59 >
< 10.0e+10 , 3 >
< ; , 78 >
< a , 0 >
< = , 59 >
< 10e-123 , 3 >
< ; , 78 >
< a , 0 >
< = , 59 >
< 10e+123 , 3 >
< ; , 78 >
< a , 0 >
< = , 59 >
< 123456 , 3 >
< ; , 78 >
< long , 7 >
< b , 0 >
< == , 46 >
< 0x8 , 3 >
< ; , 78 >
< double , 9 >
< c , 0 >
< >>= , 68 >
< 0b101 , 3 >
< ; , 78 >
< int , 6 >
< d , 0 >
< = , 59 >
< a , 0 >
< + , 37 >
< b , 0 >
< ; , 78 >
< char , 10 >
< q , 0 >
< = , 59 >
< ' , 76 >
< a , 1 >
< ' , 76 >
< ; , 78 >
< string , 0 >
< asd_a12 , 0 >
< = , 59 >
< " , 77 >
< asdads , 2 >
< " , 77 >
< ; , 78 >
< scanf , 0 >
< ( , 70 >
< " , 77 >
< %d , 2 >
< " , 77 >
< , , 79 >
< & , 53 >
< a , 0 >
< ) , 71 >
< ; , 78 >
< printf , 0 >
< ( , 70 >
< " , 77 >
< %d , 2 >
< " , 77 >
< , , 79 >
< a , 0 >
< ) , 71 >
< ; , 78 >
< return , 35 >
< 0 , 3 >
< ; , 78 >
< } , 75 >
```

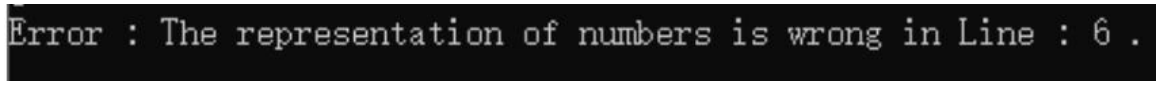
错误用例使用下列源程序进行说明：

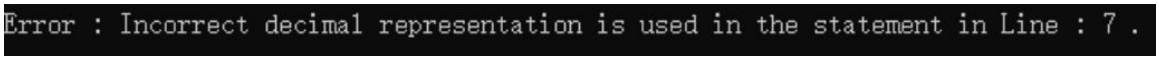
测试时要将除了测试行以外的错误行代码全注释

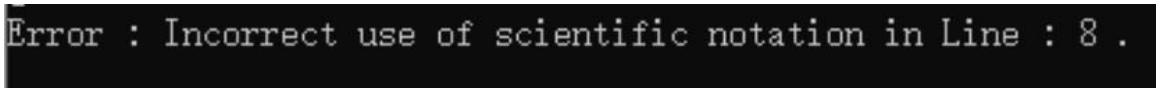
```
int main()
{
错误检测 //error1 使用不可识别字符
```

```
char b = 'asd';//error2 字符表示却有一个以上字符
a --=- b;//error3 使用不能识别符号--=-
a = 000;//error4 不合法数字
a = 12..;//error6 不合法小数
int a = 1e;//error7 不合法科学计数
}
```









error5 为不可识别 code 编码，用于防止意外情况发生