第一次实验 词法分析实验报告

设计思想

词法分析的主要任务是根据文法的词汇表以及对应约定的编码进行一定的识别，找出文件中所有的合法的单词，并给出一定的信息作为最后的结果，用于后续语法分析程序的使用；本实验针对 `PL/0 语言` 的文法、词汇表编写一个词法分析程序，对于每个单词根据词汇表输出：`(单词种类，单词的值)` 二元对。

词汇表：

种别编码	单词符号	助记符
0	begin	beginsym
1	call	callsym
2	const	dosym
3	do	endsym
4	end	ifsym
5	if	oddsym
6	odd	proceduresym
7	procedure	readsym
8	read	thensym
9	then	varsym
10	var	whilesym
11	while	wirtesym
12	write	plus
13	+	minus
14	-	times
15	*	slash
16	/	eql
17	=	neq
18	<>	lss
19	<	leq
20	<=	gtr

种别编码	单词符号	助记符
21	>	geq
22	>=	becomes
23	:=	lparen
24	(	rparen
25	)	comma
26	,	semicolon
27	;	period

除此之外的满足 `<标识符> ::=<字母>{<字母>|<数字>}` 的属于 `标识符(ident)` ； 满足 `<无符号整数> ::= <数字>{<数字>}` 属于 `常数(number)` 。

根据词法分析的一般步骤，首先分析词汇表，可以得到正规集，产生对应的正规式，然后构造出易于理解设计的NFA ，之后进行NFA向DFA的转化，最后最小化DFA就可以得到易于开发者实现的词法分析程序了。

正规式

由词汇表以及PL/0 语言的文法可以得到一组这样的正规式：

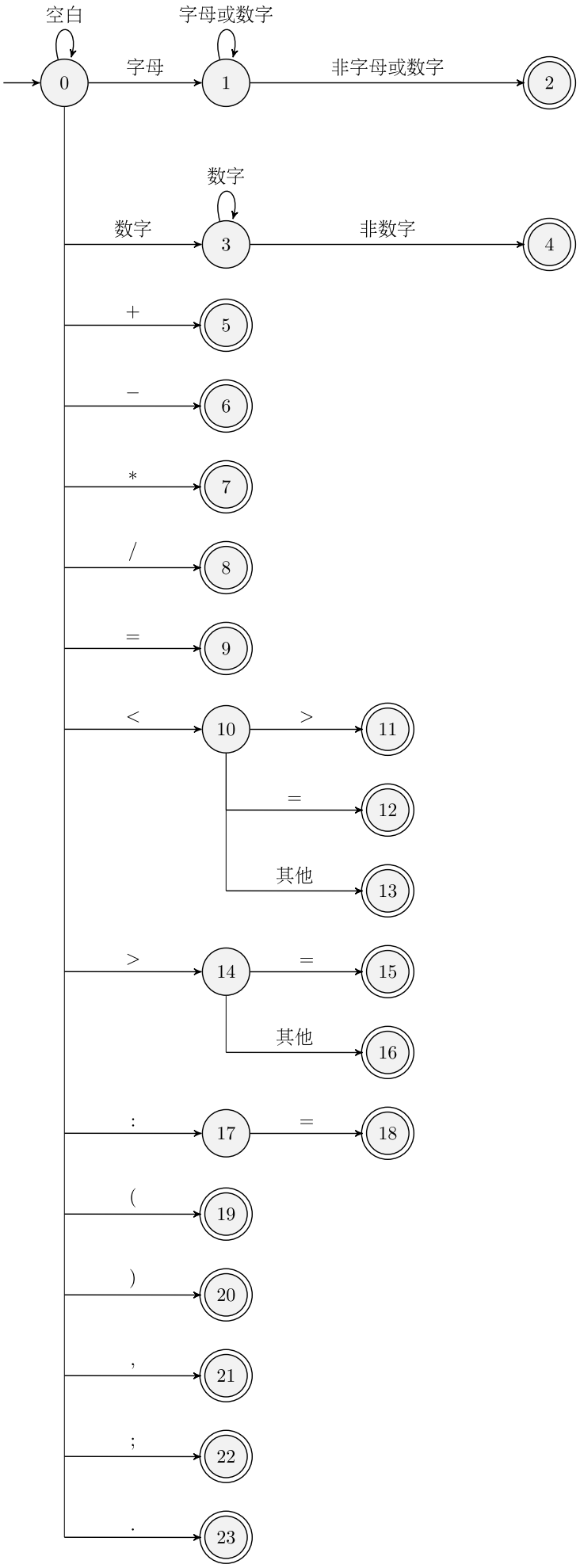
```
letter -> a|b|c|.....|z|A|B|....|Z
digit -> 0|1|2|...|9
begin
call
...
write
+
...
;
```

这样的正规式所推导出的状态转换图虽然易于理解，但是因为规则种类繁多，在实现代码时会造成代码量很大，同时对于每一条自动机上走向终态的通路存在大量重复性的代码，所以，可以在识别过程中简化一下：构造出一个可以识别 单词的自动机，然后对于识别到的单词词汇表，词汇表存在的即返回规定的助记符，不存在的不是字母数字构成的标识符或者纯数字的常数外，就判定出错单词即可，，这样最后的正规式可以写成：

```
letter -> a|b|c|.....|z|A|B|....|Z
digit -> 0|1|2|...|9
letter(letter|digit)*
digit(digit)*
各种算符：+|*|.....|.|;
```

NFA和DFA

根据词汇表和正规式可以构造出NFA以及进行转化和最小化的DFA：



(a) 状态转换图

Figure 1: 状态转换图

状态转化矩阵

根据所构造的DFA可以简单的完成词法分析代码的编写，但是这样直接根据DFA来编写的代码存在大量的分支，对于每一条从初态到终态的路径就表示识别到一个单词，就要编写相应的代码，使用到了大量的分支结构，这样的代码任务量大，易于出现逻辑上的漏洞，导致编写出的代码可能在调试中耗费大量时间，同时，当更换词汇表时，这样的程序就没有的复用的价值，所有的分析代码都要根据新的DFA来重新编写，所以为了使词法分析分析程序一般化，可以使用状态转化矩阵来控制状态转化图中状态的转化来代替各种分支判断结构，这样使得词法分析程序的控制程序和内容分离，对于新的文法，仅需构造新的状态转化矩阵即可，代码的其他控制部分无需修改。

定义 `stateTrans[i][j]` 表示 **当前状态为i，读入的字符为j** 时的下一个状态的编号，根据这个定义以及上面的状态转换图可以轻松的给出该文法的一个状态转换矩阵：

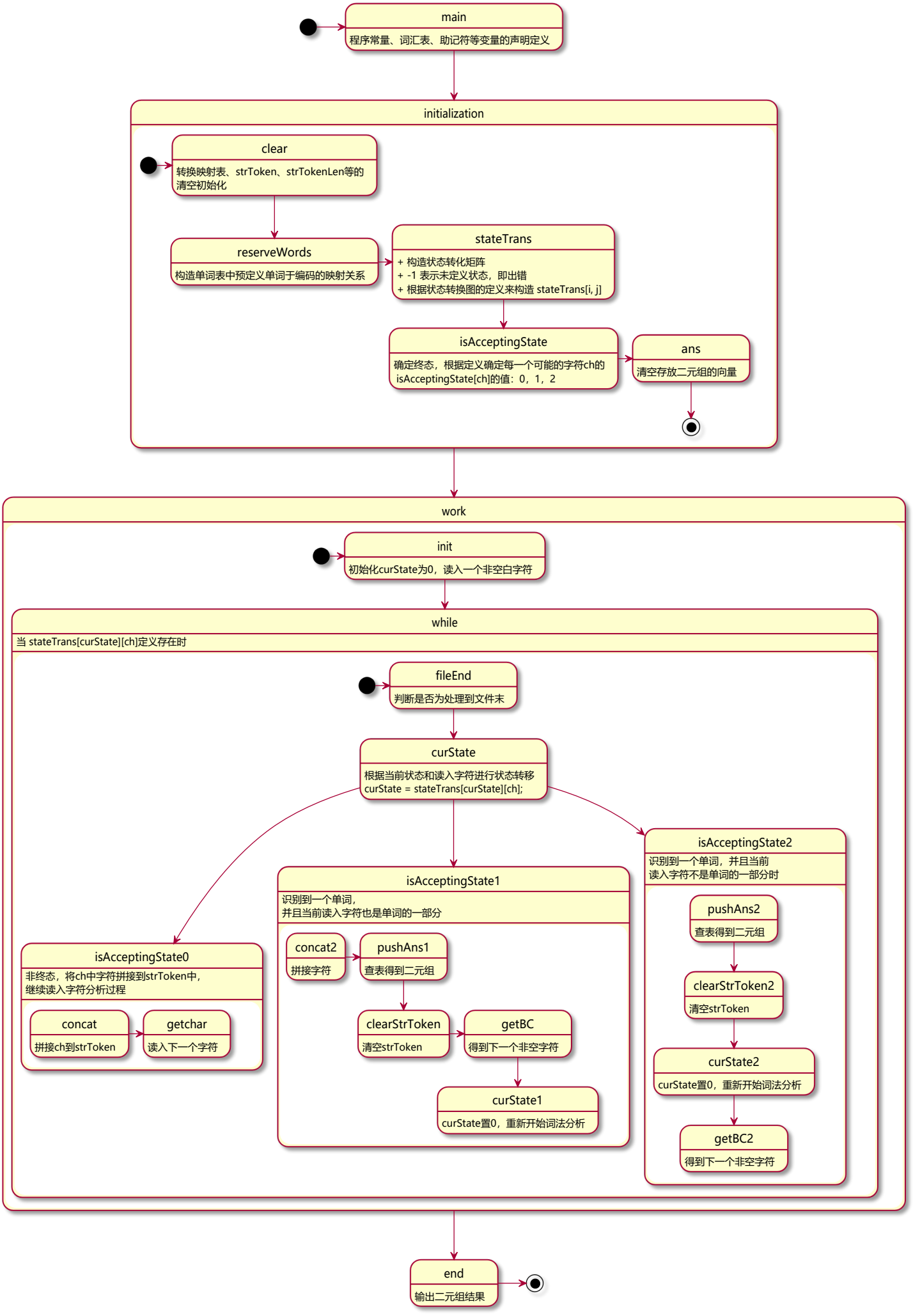
- 首先定义所有状态为未定义，值为-1
- 因为状态0是起始状态，所以要定义对应的读入字符的下一个状态，如：空白字符还是0状态、字母都是状态1、数字都是状态3其他标点字符为DFA中对应的指向的状态
- 对于状态1，根据DFA，有所有非数字、字母字符时表示得到一个单词的情况，也就是状态2，所有先置所有可能的从状态1出发的下一状态为2： `stateTrans[1,i]=2` ，然后对于字母、数字加入一个指向自身状态的情况，表示可以连续的识别一个字母开头的包含字母或数字的单词
- 状态3，读入非数字时进入状态4，数字还是状态3，类似上一个操作
- 其他的状态根据此来进行初始化即可
- 最后，为了程序的实现的更加的方便，同时为了处理某些单词的识别是靠当前读入字符便可识别的以及某些单词的识别依靠下一字符来识别的不同情况，定义一个 `isAcceptingState[i]` 终态数组，其中值得含义见代码。

这样初始化后便可以进行单词的分析了，状态转换矩阵本质上就是一个自动机的图的一个邻接矩阵， `stateTrans[i, j]` 就表示节点i的一条出边边权为j所指向的节点的标号，转化成图后就将具体的每一个判断抽象出来，由一个共用的代码块实现分析的过程：

```
curState = 初态;
GetChar();
while(stateTrans[curState][ch]有定义){
    //存在后继状态，读入、拼接
    Concat();
    //转化入下一状态，读入新字符
    curState = stateTrans[curState][ch];
    if(curState是终态){
        查找词汇表，得到二元组等操作。
    }
    Getchar();
}
```

当然其中的一些细节会不同。

算法流程


main
程序常量、词汇表、助记符等变量的声明定义
initialization
clear
转换映射表、strToken、strTokenLen等的清空初始化
reserveWords
构造单词表中预定义单词于编码的映射关系
stateTrans
+ 构造状态转化矩阵
+ -1 表示未定义状态，即出错
+ 根据状态转换图的定义来构造 stateTrans[i, j]
isAcceptingState
确定终态，根据定义确定每一个可能的字符ch的isAcceptingState[ch]的值：0，1，2
ans
清空存放二元组的向量
work
init
初始化curState为0，读入一个非空白字符
while
当 stateTrans[curState][ch]定义存在时
fileEnd
判断是否为处理到文件末
curState
根据当前状态和读入字符进行状态转移
curState = stateTrans[curState][ch];
isAcceptingState0
非终态，将ch中字符拼接到strToken中，继续读入字符分析过程
concat
拼接ch到strToken
getchar
读入下一个字符
isAcceptingState1
识别到一个单词，并且当前读入字符也是单词的一部分
concat2
拼接字符
pushAns1
查表得到二元组
clearStrToken
清空strToken
getBC
得到下一个非空字符
curState1
curState置0，重新开始词法分析
isAcceptingState2
识别到一个单词，并且当前读入字符不是单词的一部分时
pushAns2
查表得到二元组
clearStrToken2
清空strToken
curState2
curState置0，重新开始词法分析
getBC2
得到下一个非空字符
end
输出二元组结果

源程序

```
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
const int maxn = 1 << 7;
const int maxm = 1e2 + 5;
const int mod = 1e9 + 7;
const int inf = 0x3f3f3f3f;
//词汇表
const string words[] = {"begin", "call", "const", "do", "end", "if", "odd", "procedure", "read", "then",
                        "+", "-", "*", "/", "=", "<>", "<", "<=", ">", ">=", ":=",
                        "(", ")", ",", ";", "."};
//对应的助记符
const string codes[] = {"beginsym", "callsym", "constsym", "dosym", "endsym", "ifsym", "oddsym", "procedu
                        "plus", "minus", "times", "slash", "eql", "neq", "lss", "leq", "gtr", "geq", "bec
                        "lparen", "rparen", "comma", "semicolon", "period"};

char ch;                                                    //当前读入的字符
char strToken[maxn];                                        //当前读入的单词串
int strTokenLen;                                            //单词串的长度

map<string, int> symbolTable;                               //符号表，此处因为没有输出该项，所以没有使用
map<string, int> constTable;                                //常数表，此处因为没有输出该项，所以没有使用
map<string, pair<int, string> > reserveWords;   //单词和对应助记符的一个映射表，当发现单词
//时，将strToken中保存的单词在词汇表查询，存在即返回对应的助记符，不存在既是标识符或常数

//状态转换矩阵，stateTrans[i, j]表示当前在状态i，读入字符为j时下一个状态的编号
int stateTrans[maxm][maxn];
//是否是终态数组，isAcceptingState[i]表示状态i是否是终态
int isAcceptingState[maxm];
int curState;                                               //当前的状态编号

vector<pair<string, string> > ans;                          //最后分析的结果

bool IsLetter(){                                            //判断一个字符是否为字母
    if((ch >= 'a' && ch <'z') || (ch >= 'A') && (ch <= 'Z'))return true;
    return false;
}
bool IsDigital(){                                           //判断数字
    if(ch >= '0' && ch <= '9')return true;
    return false;
}
bool IsBlank(){                                             //判断是否为空白字符
    if(ch == ' ' || ch == '\n' || ch == '\r' || ch == '\t')return true;
    return false;
}
bool FileEnd;                                               //是否读到文件末

//读入一个字符到ch，当读到文件末是scanf返回-1，此时FileEnd的值就为假False
void GetChar(){
    FileEnd = ~scanf("%c", &ch);
    // ch = getchar();
}
```

```
void GetBC(){                                                    //跳过空白符
    GetChar();
    while(FileEnd && IsBlank())GetChar();
}
void Concat(){                                                   //将ch加入到strToken中
    if(strTokenLen < maxn)strToken[strTokenLen++] = ch;
}
//根据词汇表的映射返回当前识别到的单词的助记符
pair<int, string> Reserve(){
    string s = string(strToken);
    if(reserveWords.count(s))return reserveWords[s];
    else if(curState == 2)return make_pair(0, "ident");
    else return make_pair(0, "number");
    return make_pair(0, "");
}
void pushAns(){                                                  //增加结果二元组
    ans.push_back(make_pair(string(strToken), Reserve().second));
}
void InsertId(){                                                 //将识别到的单词插入符号表（此程序未使用）
    symbolTable[string(strToken)] = symbolTable.size() + 1;
}
void InsertConst(){                                              //将识别到的常数插入常数表（此程序未使用）
    constTable[string(strToken)] = constTable.size() + 1;
}

//初始化函数，除了各变量的置空初始化外，根据不同文法的DFA初始化状态转化矩阵
void init(){
    FileEnd = true;
    ch = ' ';
    symbolTable.clear();
    constTable.clear();
    reserveWords.clear();
    memset(strToken, '\0', sizeof strToken);
    strTokenLen = 0;

    //构造单词表中预定义单词于编码的映射关系
    int len = sizeof(words) / sizeof(words[0]);
    for(int i = 0; i < len; ++i)reserveWords[words[i]] = make_pair(i, codes[i]);

    //构造状态转化矩阵
    //-1 表示未定义状态，即出错
    //inf表示终态，表示识别到一个单词（使用isaccepting来表示
    memset(stateTrans, -1, sizeof stateTrans);
    //对于状态0，读入空白仍为该状态，字母进入状态1，数字进入状态2等等
    stateTrans[0][' '] = stateTrans[0]['\n'] = stateTrans[0]['\r'] = stateTrans[0]['t'] = 0;
    for(int i = 'a'; i <= 'z'; ++i)stateTrans[0][i] = 1;
    for(int i = 'A'; i <= 'Z'; ++i)stateTrans[0][i] = 1;
    for(int i = '0'; i <= '9'; ++i)stateTrans[0][i] = 3;
    stateTrans[0]['+'] = 5;
    stateTrans[0]['-'] = 6;
    stateTrans[0]['*'] = 7;
    stateTrans[0]['/'] = 8;
    stateTrans[0]['='] = 9;
    stateTrans[0]['<'] = 10;
    stateTrans[0]['>'] = 14;
    stateTrans[0][':'] = 17;
    stateTrans[0]['('] = 19;
    stateTrans[0][')'] = 20;
    stateTrans[0][','] = 21;
    stateTrans[0][';'] = 22;
```

```
    stateTrans[0]['.'] = 23;

    //对其他状态定义：
    //1:
    for(int i = 0; i < maxn; ++i)stateTrans[1][i] = 2;
    for(int i = 'a'; i <= 'z'; ++i)stateTrans[1][i] = 1;
    for(int i = 'A'; i <= 'Z'; ++i)stateTrans[1][i] = 1;
    for(int i = '0'; i <= '9'; ++i)stateTrans[1][i] = 1;

    //3:
    for(int i = 0; i < maxn; ++i)stateTrans[3][i] = 4;
    for(int i = '0'; i <= '9'; ++i)stateTrans[3][i] = 3;

    //10:
    fill(stateTrans[10], stateTrans[10] + maxn, 13);
    stateTrans[10]['>'] = 11;
    stateTrans[10]['='] = 12;

    //14:
    fill(stateTrans[14], stateTrans[14] + maxn, 13);
    stateTrans[14]['='] = 15;

    //17:
    stateTrans[17]['='] = 18;

    //确定终态：
    //0:表示非终态
    //1:表示根据当前读入的字符拼接到strToken后即为一个单词
        //（显然这样下一次单词分析需要再读入新字符）
    //2:表示根据当前读入字符可以判断出strToken中为一个单词
        //（显然此时读入的字符要归入到下一次单词分析）
    fill(isAcceptingState, isAcceptingState + maxm, 1);
    isAcceptingState[0] = isAcceptingState[1] = isAcceptingState[3] = isAcceptingState[10] = isAcceptingS
    isAcceptingState[2] = isAcceptingState[4] = isAcceptingState[13] = isAcceptingState[16] = 2;

    ans.clear();
}


void work(){                                    //词法分析一般控制过程
    curState = 0;
    GetBC();
    while(~stateTrans[curState][ch]){           //当当前的状态合法时进行分析
        if(!FileEnd)ch = '\0';                  //如果是读到文件末，对最后遗留在strToken进行分析后退出子程序
        curState = stateTrans[curState][ch];    //根据当前状态和读入字符进行状态转移
        if(isAcceptingState[curState] == 0){    //非终态，将ch中字符拼接到strToken中，继续读入字符分析过程
            Concat();
            GetChar();
        }
        else if(isAcceptingState[curState] == 1){//识别到一个单词，并且当前读入字符也是单词的一部分
            Concat();                           //将读入字符ch拼接
            cerr << "1.find a words: " << strTokenLen << ": " << strToken << endl;
            pushAns();                          //调用保存结果函数，查表等获得二元组
            memset(strToken, '\0', sizeof strToken);//清空strToken等，为下一次分析做准备
            strTokenLen = 0;
            GetBC();                            //读到下一个非空字符
            curState = 0;
        }
        else if(isAcceptingState[curState] == 2){//识别到一个单词，并且当前读入字符不是单词的一部分时
            cerr << "2.find a words: " << strTokenLen << ": " << strToken << endl;
            pushAns();
```

```
            memset(strToken, '\0', sizeof strToken);
            strTokenLen = 0;
            // Concat();
            //当前字符要进入下一次分析，所以不拼接到strToken中，也不进行读入新字符的操作（除空白字符外）
            curState = 0;
            if(IsBlank())GetBC();       //如果当前读入的字符是空白符，也就是用空白符分隔所得到的单词时，
                                        //显然为了下一次分析要不断地读到非空字符
        }
        else{                           //未定义的状态，此时读入的字符是文法所为定义的字符，
                                        //提示报错，退出程序
            cerr << "error!" << endl;
            break;
        }
        if(ch == '\0')break;            //分析到文件末结束分析
    }
}


int main(){

    // freopen("test.txt", "r", stdin);
    // freopen("ans.txt", "w", stdout);

    init();
    work();
    // for(auto i: ans)cout << "(" << i.second << "," << i.first << ")" << endl;
        //输出二元组结果
    for(int i = 0; i < ans.size(); ++i)cout << "(" << ans[i].second << "," << ans[i].first << ")" << endl

    return 0;
}
```

调试数据

input.txt:（这里因为在运行时，当输入完数据后要手动输入一个 `Ctrl Z` 表示输入结束）

```
const a=10;
var b,c;
begin
read(b);
c:=a+b;
write(c)
end.^Z
```

output.txt:（此处的标准输出即为二元组对，标准错误输出流中是每一次到达自动机终态时识别到的单词）

```
get a words: "const"
get a words: "a"
get a words: "="
get a words: "10"
get a words: ";"
get a words: "var"
get a words: "b"
get a words: ","
get a words: "c"
get a words: ";"
get a words: "begin"
get a words: "read"
get a words: "("
get a words: "b"
get a words: ")"
get a words: ";"
get a words: "c"
get a words: ":="
get a words: "a"
get a words: "+"
get a words: "b"
get a words: ";"
get a words: "write"
get a words: "("
get a words: "c"
get a words: ")"
get a words: "end"
get a words: "."
(constsym,const)
(ident,a)
(eql,=)
(number,10)
(semicolon,;)
(varsym,var)
(ident,b)
(comma,,)
(ident,c)
(semicolon,;)
(beginsym,begin)
(readsym,read)
(lparen,()
(ident,b)
(rparen,))
(semicolon,;)
(ident,c)
(becomes,:=)
(ident,a)
(plus,+)
(ident,b)
(semicolon,;)
(writesym,write)
(lparen,()
(ident,c)
(rparen,))
(endsym,end)
(period,.)
```

体会

对 `PL/0 语言` 的词法分析，因为其文法较为简单，所以对应的词法分析过程易于实现，在得到词法分析的DFA之后，可以直接进行程序代码的编写，这样虽然可以实现分析过程，但是这样实现的词法分析程序很大一部分代码不能重用，所以为了实现一个词法分析程序的一般形式，故根据DFA推导出对应的状态转换矩阵，然后使用一个一般化的框架来实现分析过程，这样一个过程其实就是将DFA中各状态之间的转换关系用一个邻接矩阵来描述，将具体的每一个分析的过程变成图的形式，简化代码的同时大大增加代码的利用率，在更换文法以及对应的词汇表后，只需重新分析构造状态转换矩阵并替换即可实现新的词法分析器。在实现的过程中，发现某些分析的终止状态是与当前字符有关，所以要单独处理这一种情况，于是添加了isAcceptingState数组来记录所有的终态的类别，使得在一般化的状态转换函数中的逻辑更加的清晰，这也给我启示，在实现一个一般化的代码时，切忌生搬硬套已经给出的伪码，教条主义不可取，要理论与实践的结合，根据自己当前任务结合课本知识进行改造利用才能完成任务。

HTML