一、实验基础信息

> 个人信息

201700130011 —— 刘建东 —— 17级菁英班

> 实验信息

日期：2020.4.18
题目：PL/0 语言词法分析（GETSYM）

二、PL/0 语言文法 BNF

```
〈程序〉→〈分程序〉.
〈分程序〉→ [<常量说明部分>][<变量说明部分>][<过程说明部分>]〈语句〉
<常量说明部分> → CONST<常量定义>{ ,<常量定义>};
<常量定义> → <标识符>=<无符号整数>
<无符号整数> → <数字>{<数字>}
<变量说明部分> → VAR<标识符>{ ,<标识符>};
<标识符> → <字母>{<字母>|<数字>}
<过程说明部分> → <过程首部><分程序>; {<过程说明部分>}
<过程首部> → procedure<标识符>;
<语句> → <赋值语句>|<条件语句>|<当型循环语句>|<过程调用语句>|<读语
          句>|<写语句>|<复合语句>|<空>
<赋值语句> → <标识符>:=<表达式>
<复合语句> → begin<语句>{ ; <语句>}end
<条件> → <表达式><关系运算符><表达式>|odd<表达式>
<表达式> → [+|-]<项>{<加减运算符><项>}
<项> → <因子>{<乘除运算符><因子>}
<因子> → <标识符>|<无符号整数>|(<表达式>)
<加减运符> → +|-
<乘除运算符> → *|/
<关系运算符> → =|#|<|<=|>|>=
<条件语句> → if<条件>then<语句>
<过程调用语句> → call<标识符>
<当型循环语句> → while<条件>do<语句>
<读语句> → read(<标识符>{ , <标识符>})
<写语句> → write(<标识符>{, <标识符>})
<字母> → a|b|c…x|y|z
<数字> → 0|1|2…7|8|9
```

三、词法分析过程

在词法分析中，我们主要需要识别如下三类单词：

1. 关键字
2. 标识符（包括变量、函数名等）
3. 常量

首先对于关键字，我们可以观察 BNF 进行关键字的罗列。

```
string keywords[30] = {".", "CONST", ",", ";", "=", "VAR", "procedure",
                       ":=", "begin", "end", "odd", "+", "-", "*", "/", "=",
                       "#", "<", "<=", ">", ">=", "if", "then", "call",
                       "while", "do", "read", "write", "(", ")"};
```

其次对于标识符，我们可以观察 BNF 发现，标识符由字母开头，后续跟上任意个字母或数字。而对于常量则完全由数字组成。

因此我们便有了以下策略：

1. 如果当前字符是换行或空格，则直接跳过
2. 如果当前字符为数字，则不断读取下一个字符，直到下一个字符不为数字
3. 如果当前字符为字母，则不断读取下一个字符，直到下一个字符不为数字或字符
    - 读完以后需要判断读取的字符串是否为关键字
    - 可以使用 STL 中的 map 直接判断 O(logn)，也可以直接遍历，O(n)
4. 对于其他字符，则一直读取，直到出现空格或换行
    - 读完以后需要判断读取的字符串是否为关键字，如果不是，则报错

有了读取策略后，我们需要制定一个存储的方法，维护三个表，可以用 链表 / 数组 / 队列 实现：

1. SYM：表示当前单词的类别
2. ID：如果当前单词是标识符，则将其对应的值存储到 ID 中
3. NUM：如果当前单词是常量，则将其对应的值存储到 NUM 中

有了存储、识别策略后，我们便可以完成词法分析的过程。

四、运行结果

此处由于篇幅的原因，我们将给出一小段 PL/0 文法的程序及其对应的词法分析结果。

PL/0 文法程序

```
var x, y, z, q, r, n, f;

procedure multiply;
var a, b;
```

```
begin
  a := x;
  b := y;
  z := x*y
end;
```

词法分析结果

```
var
x
,
y
,
z
,
q
,
r
,
n
,
f
;
procedure
multiply
;
var
a
,
b
;
begin
a
:=
x
;
b
:=
y
;
z
:=
x
*
y
end
```

```
;
```

五、源代码

```
#include <iostream>
#include <map>
#include <vector>
#include <string>
using namespace std;

string keywords[30] = {".", "CONST", ",", ";", "=", "VAR", "procedure", ":=", "be
gin", "end", "odd", "+", "-", "*", "/", "=", "#", "<", "<=", ">", ">=", "if", "th
en", "call", "while", "do", "read", "write", "(", ")"};
const int keytype = 29, tagtype = 30, numtype = 31;

vector<int> SYM;
vector<string> ID;
vector<long long> NUM;
map<string, int> mp;

void init(char *indir, char *outdir) {
    freopen(indir, "r", stdin);
    freopen(outdir, "w", stdout);
    mp.clear(); SYM.clear(); ID.clear(); NUM.clear();
    for(int i = 0; i <= 29; i++)
        mp[keywords[i]] = i;
}

void output(){
    int p2 = 0, p3 = 0, sz = SYM.size();
    for(int p1 = 0; p1 < sz; p1++){
        if(SYM[p1] <= keytype) cout << keywords[SYM[p1]] << endl;
        else if(SYM[p1] == tagtype) cout << ID[p2++] << endl;
        else cout << NUM[p3++] << endl;
    }
}

bool GETSYM(char *indir, char *outdir) {
    init(indir, outdir);
    string line;
    while(getline(cin, line)){
        for(int i = 0; line[i]; i++){
            if(line[i] == ' ' || line[i] == '\n' || (int)line[i] == 13){
                continue;
            }
            else if(line[i] >= '0' && line[i] <= '9'){
```

```
                // identify number
                long long number = line[i]-'0';
                while(line[i+1] && line[i+1] >= '0' && line[i+1] <= '9'){
                    number = number * 10 + (int)(line[++i] - '0');
                }
                SYM.push_back(numtype);
                NUM.push_back(number);
            }
            else if(line[i] >= 'a' && line[i] <= 'z'){
                // identify variable
                string tmp = "\0";
                tmp += line[i];
                while(line[i+1] && ((line[i+1] >= '0' && line[i+1] <= '9') || (li
ne[i+1] >= 'a' && line[i+1] <= 'z'))){
                    tmp += line[++i];
                }
                if(mp.find(tmp) != mp.end()) SYM.push_back(mp[tmp]);
                else SYM.push_back(tagtype), ID.push_back(tmp);
            }
            else{
                string tmp = "\0";
                tmp += line[i];
                while(mp.find(tmp) == mp.end() && line[i+1] && line[i+1] != ' ' &
& line[i+1] != '\n' && (int)line[i+1] != 13){
                    tmp += line[++i];
                }
                if(mp.find(tmp) != mp.end()) SYM.push_back(mp[tmp]);
                else return 0;
            }
        }
    }
    output();
    return 1;
}

int main() {
    GETSYM(indir, outdir);
    return 0;
}
```