```
PASCAL 语言子集(PL/0)词法分析器的设计与实现
PL/0 语言的 BNF 描述(扩充的巴克斯范式表示法)
\langle block \rangle \rightarrow [\langle condec1 \rangle][\langle vardec1 \rangle][\langle proc \rangle] \langle body \rangle
\langle \text{condecl} \rangle \rightarrow \text{const} \langle \text{const} \rangle \{, \langle \text{const} \rangle \};
⟨const⟩ → ⟨id⟩:=⟨integer⟩
\langle \text{vardecl} \rangle \rightarrow \text{var} \langle \text{id} \rangle \{, \langle \text{id} \rangle \};
\langle proc \rangle \rightarrow procedure \langle id \rangle ([\langle id \rangle \{, \langle id \rangle] \}) ; \langle block \rangle \{; \langle proc \rangle \}
⟨body⟩ → begin ⟨statement⟩{;⟨statement⟩} end
\langle \text{statement} \rangle \rightarrow \langle \text{id} \rangle := \langle \text{exp} \rangle
                         | if <lexp> then <statement>[else <statement>]
                         |while <lexp> do <statement>
                         |call \langle id \rangle ([\langle exp \rangle \{, \langle exp \rangle \}])
                         <body>
                         | read (\langle id \rangle \{, \langle id \rangle \}) |
                         |\text{write }(\langle \exp \rangle \{, \langle \exp \rangle \})|
\langle lexp \rangle \rightarrow \langle exp \rangle \langle lop \rangle \langle exp \rangle | odd \langle exp \rangle
\langle \exp \rangle \rightarrow [+|-] \langle \operatorname{term} \rangle \{\langle \operatorname{aop} \rangle \langle \operatorname{term} \rangle \}
<term> → <factor>{<mop><factor>}
\langle factor \rangle \rightarrow \langle id \rangle | \langle integer \rangle | (\langle exp \rangle)
\langle 1_{\text{op}} \rangle \rightarrow = |\langle \rangle |\langle |\langle =| \rangle |\rangle =
\langle aop \rangle \rightarrow + | -
\langle mop \rangle \rightarrow * | /
\langle id \rangle \rightarrow 1\{1|d\}
                             (注: 1表示字母)
\langle integer \rangle \rightarrow d\{d\}
注释:
⟨prog⟩: 程序 ; ⟨block⟩: 块、程序体 ; ⟨condecl⟩: 常量说明 ; ⟨const⟩:
常量:
语句:
<exp>: 表达式; <lexp>: 条件; <term>: 项; <factor>: 因子; <aop>:
加法运算符:
〈mop〉: 乘法运算符; 〈lop〉: 关系运算符
odd: 判断表达式的奇偶性。
program <id>;
<br/><block> -> [<condecl>][<vardecl>][<proc>]<body>
const a:=5, const, b:=6;
<vardecl>->var <id>,<id>,<id>;
<jd> ( [<id>,<id>,<id>]) ;<block>
<body>->
begin
     <statement>;
     <statement>;
```

```
end
<statement>-> <id>:=<exp>
   | if <lexp> then <statement>[else< statement>]
   | while <lexp> do <statement>
   | call <id> ( [<exp>,<exp>,<exp>])
   | <body>
   | read (<id>,<id>,<id>)
   | write (<exp>,<exp>)
<le>xp>-> <exp><lop><exp> | odd <exp>
<exp> -> [+|-] <term><aop><term><aop><term>
<term>-> <factor> <mop><factor> <mop><factor>
<lop>-> =| < >|<|<=|>|>=
 <id>->字母开头的数字字母组合
keyword:
program
const
var
procedure
begin
if then else while call read write odd
end
id:
字母开头的数字字母组合
symbol:
+-*/ < > <>
```

LIT 0 a	将常数值取到栈顶,a为常数值
LOD 1 a	将变量值取到栈顶,a为偏移量,1为层差
STO 1 a	将栈顶内容送入某变量单元中,a为偏移量,1为层差
CAL 1 a	调用过程,a为过程地址,1为层差
INT 0 a	在运行栈中为被调用的过程开辟a个单元的数据区
JMP 0 a	无条件跳转至a地址
JPC 0 a	条件跳转,当栈顶布尔值非真则跳转至a地址,否则顺
	序执行
OPR 0 0	过程调用结束后,返回调用点并退栈
OPR 0 1	栈顶元素取反
OPR 0 2	次栈顶与栈顶相加,退两个栈元素,结果值进栈
OPR 0 3	次栈顶减去栈顶,退两个栈元素,结果值进栈
OPR 0 4	次栈顶乘以栈顶,退两个栈元素,结果值进栈
OPR 0 5	次栈顶除以栈顶,退两个栈元素,结果值进栈
OPR 0 6	栈顶元素的奇偶判断,结果值在栈顶
OPR 0 7	
OPR 0 8	次栈顶与栈顶是否相等,退两个栈元素,结果值进栈
OPR 0 9	次栈顶与栈顶是否不等,退两个栈元素,结果值进栈
OPR 0 10	次栈顶是否小于栈顶,退两个栈元素,结果值进栈
OPR 0 11	次栈顶是否大于等于栈顶,退两个栈元素,结果值进
	栈
OPR 0 12	次栈顶是否大于栈顶,退两个栈元素,结果值进栈
OPR 0 13	次栈顶是否小于等于栈顶,退两个栈元素,结果值进
	栈
OPR 0 14	栈顶值输出至屏幕
OPR 0 15	屏幕输出换行
OPR 0 16	从命令行读入一个输入置于栈顶