SHANGHAI UNIVERSITY

**<编译原理>实验报告**

|  |  |
| --- | --- |
| **学 院** | **计算机工程与科学学院** |
| **组 号** | **A04** |
| **实验题号** | **实验三 语法分析** |
| **姓名** | **刘洋** |
| **学号** | **16121368** |
| **日期** | **2018.4.25** |

|  |  |
| --- | --- |
| 学号 | 姓名 |
| 16121348 | 卿波 |
| 16121368 | 刘洋 |
| 16121391 | 陈杰 |
| 16122804 | 刘锦 |

## 实验目的

* 给出 PL/0 文法规范，要求编写 PL/0 语言的语法分析程序。
* 通过设计、编制、调试一个典型的语法分析程序，实现对词法分析 程序所提供的单词序列进行语法检查和结构分析，进一步掌握常用 的语法分析方法。
* 选择一种语法分析方法（递归子程序法、LL(1)分析法、算符优先分 析法、SLR(1)分析法）；选择常见程序语言都具备的语法结构，如赋 值语句，特别是表达式，作为分析对象。

## 实验内容：

* 已给 PL/0 语言文法，构造表达式部分的语法分析器。
* 分析对象的 BNF 定义如下：

<程序>::=<分程序>.

<分程序> ::=[<常量说明>][<变量说明>][<过程说明>]<语句>

<常量说明> ::=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> ::=VAR <标识符>{, <标识符>};

<标识符> ::=<字母>{<字母>|<数字>}

<过程说明> ::=<过程首部><分程序>{; <过程说明> };

<过程首部> ::=PROCEDURE <标识符>;

<语句> ::=<赋值语句>|<条件语句>|<当循环语句>|<过程调用语句>

|<复合语句>|<读语句><写语句>|<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> ::= <表达式> <关系运算符> <表达式> |ODD<表达式>

<表达式> ::= [+|-]<项>{<加法运算符> <项>}

<项> ::= <因子>{<乘法运算符> <因子>}

<因子> ::= <标识符>|<无符号整数>| ‘(’<表达式>‘)’

<加法运算符> ::= +|-

<乘法运算符> ::= \*|/

<关系运算符> ::= =|#|<|<=|>|>=

<条件语句> ::= IF <条件表达式> THEN <语句>

<过程调用语句> ::= CALL 标识符

<当循环语句> ::= WHILE <条件表达式> DO <语句>

<读语句> ::= READ‘(’<标识符>{,<标识符>}‘)’

<写语句> ::= WRITE‘(’<表达式>{,<表达式>}‘)’

<字母> ::= a|b|…|X|Y|Z

<数字> ::= 0|1|…|8|9

## 实验要求：

* 确定语法分析的方法。
* 将实验二“词法分析”的输出结果，作为表达式语法分析器的输入， 进行语法解析，对于语法正确的表达式，报告“语法正确”； 对于语法错误的表达式，报告“语法错误”， 指出错误原因。
* 源程序中字符不区分大小写，即：“a1”和“A1”是同一个标识符。
* 准备至少 10 组测试用例，每组测试用例包括：输入文件和输出结果。

## 算法描述：

本实验采用递归子程序法进行语法分析，建立在上一次实验的基础之上，输入的为PL/0 表达式源语言，经过词法分析后使用词法分析的结果进行语法分析。

首先将词法分析的结果对应的系统名称存在名为res的vector容器中，利用getSym()函数扫描得到容器中的下一个词，根据PL/0语言的EBNF描述自左到右对每一个词进行判断，判断是否满足语法要求。在复杂的语法单位中，使用递归调用，从最基础的标识符、项、因子开始写起，到表达式的判断，到各种语句，再加上常量变量的声明定义部分等，组成分程序，最后构成整个程序。以表达式判断isExpression()为例，根据表达式的语法规则，需要判断是否为项，就需要调用isTerm()函数，判断项中需要判断是否为因子，需要调用isFactor()函数，在判断因子的函数中，又需要判断是否为表达式，这是个典型的递归调用的过程。别的语法也是如此。

## 实验代码：

部分C++程序如下：

|  |
| --- |
| #include<bits/stdc++.h>  using namespace std;  class Analyzer  {  public:  string text;//PL0文本  //系统名称及对应的符号串  string clas[29]={"plus","minus","times","slash", "oddsym","eql", "neq","lss","leq","gtr","geq","lparen","rparen","comma","semicolon","period","becomes","beginsym","endsym","ifsym", "thensym", "whilesym","writesym","readsym","dosym","callsym","constsym","varsym","procsym"};  string val[29]={"+","-", "\*", "/", "odd", "=", "#", "<", "<=", ">",">=", "(",")",",",";",".",":=","begin","end","if","then","while", "write", "read","do","call","const","var","procedure"};  map<string,string> mp; //clas 与 val 的映射关系  int idx; //用于标记text的读取位置  vector<string> words; //text中的所有单词  vector<string> res; //words 所对应的系统名字  string str=""; //语法分析的最小单元  int now=0; //用于标记语法分析中所处res的位置  Analyzer(string \_text); //初始化text及mp  virtual ~Analyzer();  string getNext(); //得到text的下一个单词  string getSym(); //得到res中的下一个单词  //qb  bool isLegalName(); //判断res中的命名是否有number+ident非法组合  bool isNone(char ch); //判断是否为'/t',' ','\n',EOF此类空字符  bool isNumber(string s); //判断是否为整数  bool isIdent(string s); //判断是否为标识符  bool isExpression(); //判断是否为表达式  bool isTerm(); //判断是否为项  bool isFactor(); //判断是否为因子  //cj  bool isCondition(); //判断是否为条件表达式  bool isSentence();//判断是否为语句  bool isComplexSentence();//复合语句  bool isJudgement();//判断是否为条件语句  bool isLoop();//判断是否为循环语句  bool isAssignment(); //赋值语句  //ly  bool isProgram(); //程序  bool isBlock();//分程序  bool isProcedureDescription();//过程说明  bool isProcedureCall();//过程调用语句  bool isProcedureHead();//过程首部  //lj  bool isConstDeclaration();//常量说明  bool isVarDeclaration();//变量说明  bool isConstDefinition();//常量定义  bool isWrite();//读语句  bool isRead();//写语句  bool find(string s); //在map中查找s  protected:  private:  };  //标识符  bool Analyzer::isIdent(string s)  {  if(s==""||!(s[0]>='a'&&s[0]<='z')){  return false;  }  for(int i=1;i<s.length();i++)  {  if(!(s[i]>='a'&&s[i]<='z'||s[i]>='0'&&s[i]<='9')){  cout<<"标识符不合法"<<endl;  return false;  }  }  return true;  }  string Analyzer::getSym()  {  return now<res.size()?res[now++]:"";  }  //表达式  bool Analyzer::isExpression()  {  bool flag=true;  if(str=="plus"||str=="minus")  {  str=getSym();  }  flag&=isTerm();  while(str=="plus"||str=="minus")  {  str=getSym();  flag&=isTerm();  }  return flag;  }  //判断是否为条件表达式  bool Analyzer::isCondition()  {  bool flag=true;  if(str=="odd")  {  str=getSym();  flag&=isExpression();  }  else  {  flag&=isExpression();  if(str=="eql"||str=="neq"||str=="lss"||str=="leq"||str=="gtr"||str=="geq"){  str=getSym();  flag&=isExpression();  }  else  {  flag=false;  cout<<"条件表达式中关系运算符有错"<<endl;  str=getSym();  }  }  return flag;  }  //赋值语句  bool Analyzer::isAssignment()  {  bool flag=true;  if(str=="ident")  {  str=getSym();  if(str=="becomes")  {  str=getSym();  flag&=isExpression();  }  else  {  flag=false;  cout<<"赋值语句缺少:="<<endl;  str=getSym();  }  }  else  {  flag=false;  cout<<"赋值语句缺少标识符"<<endl;  str=getSym();  }  return flag;  }  //条件语句  bool Analyzer::isJudgement()  {  bool flag=true;  if(str=="ifsym")  {  str=getSym();  flag&=isCondition();  if(str=="thensym")  {  str=getSym();  flag&=isSentence();  }  else  {  flag=false;  cout<<"条件语句缺少then"<<endl;  str=getSym();  }  }  else{  flag=false;  cout<<"条件语句缺少if"<<endl;  str=getSym();  }  return flag;  }  //分程序  bool Analyzer::isBlock()  {  bool flag=true;  if(str=="constsym")  {  flag&=isConstDeclaration();  }  if(str=="varsym")  {  flag&=isVarDeclaration();  }  if(str=="procsym")  {  flag&=isProcedureDescription();  }  flag&=isSentence();  return flag;  } |

## 实验结果：

**1、测试样例1：**

**输入：**

var x,y,z;

begin

x := 10;

y := -5;

read(z);

if z > 3 then

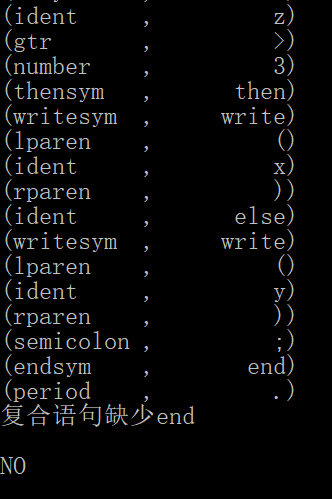
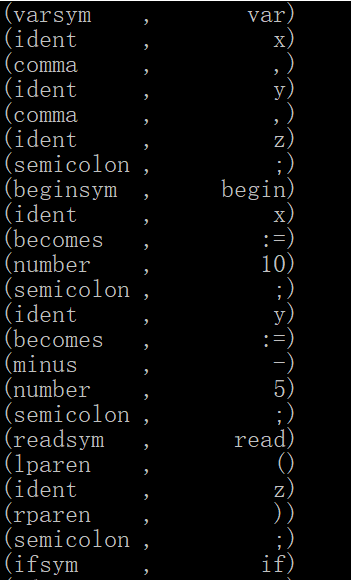
write(x)

else

write(y);

end.

**输出：**



**修改后：**

var x,y,z;

begin

x := 10;

y := -5;

read(z);

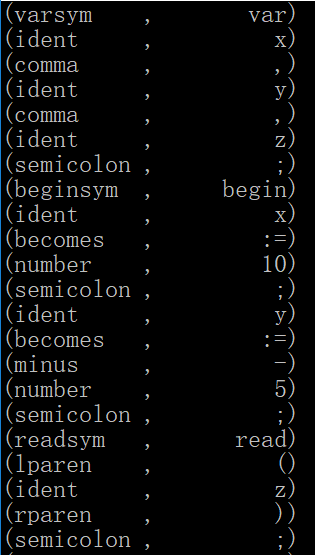
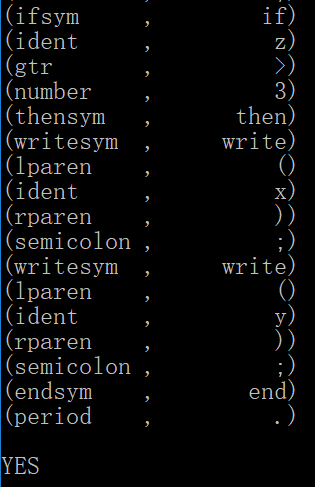
if z > 3 then

write(x);

write(y);

end.

**输出：**

**2、测试样例2：**

**输入：**

CONST A=10;

VAR B,C;

PROCEDURE P;

VAR D;

PROCEDURE Q;

VAR X;

BEGIN

READ(X);

D:=X;

WHILE X#0

DO CALL P;

END;

BEGIN

WRITE(D);

CALL Q;

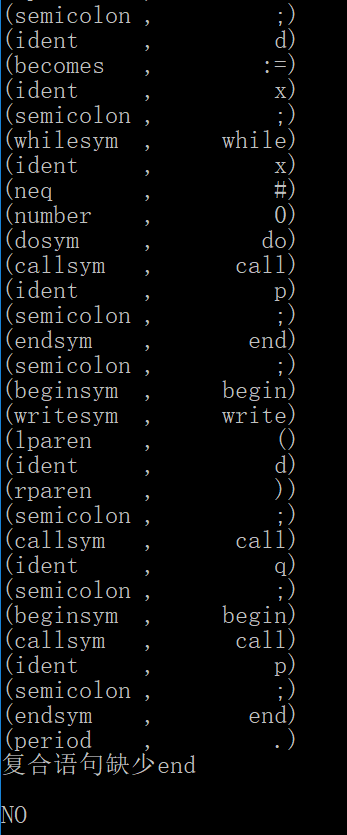
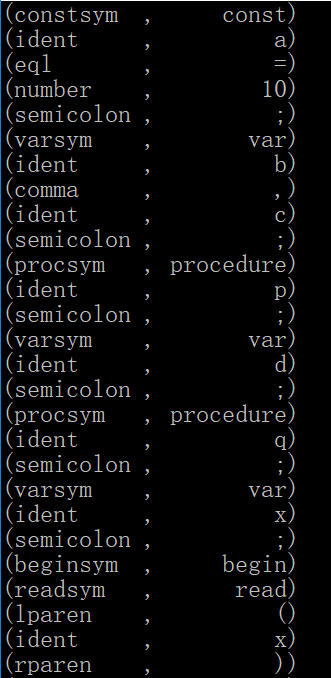
END

BEGIN

CALL P;

END..

**输出：**



**修改后：**

CONST A=10;

VAR B,C;

PROCEDURE P;

VAR D;

PROCEDURE Q;

VAR X;

BEGIN

READ(X);

D:=X;

WHILE X#0

DO CALL P;

END;

BEGIN

WRITE(D);

CALL Q;

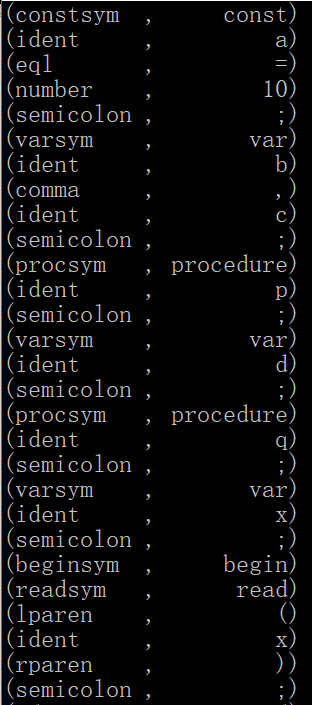
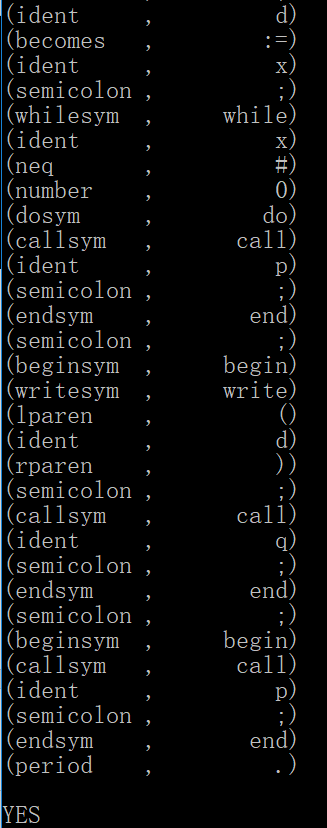
END;

BEGIN

CALL P;

END.

**输出：**

**3、测试样例3：**

**输入：**

var

b+c;

begin

read(b);

while b # 0 do

begin

call p;

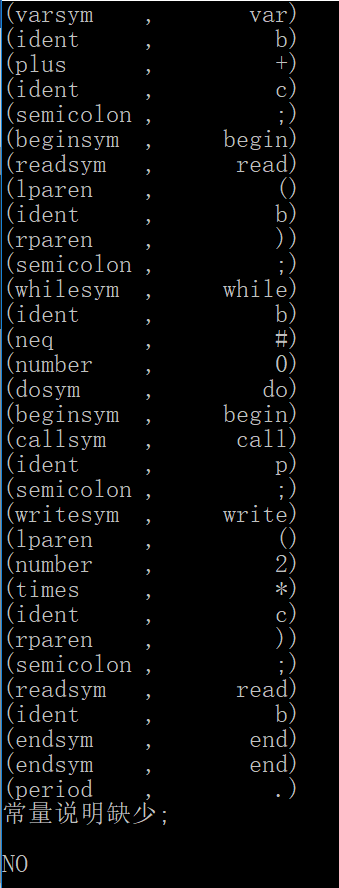
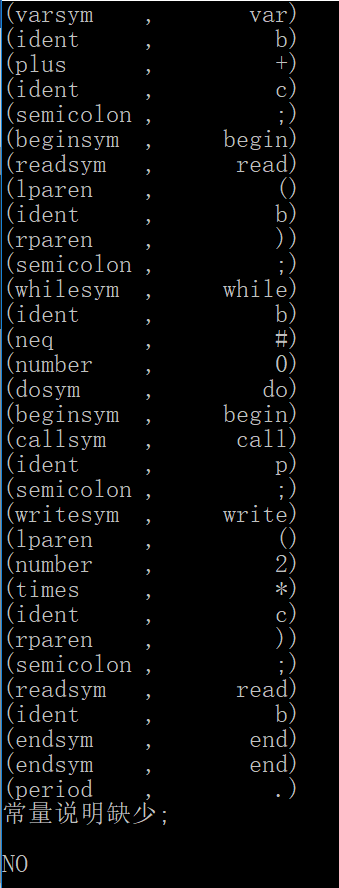
write(2 \* c);

read b

end

end.

**输出：**



**修改后：**

var

b,c;

begin

read(b);

while b # 0 do

begin

call p;

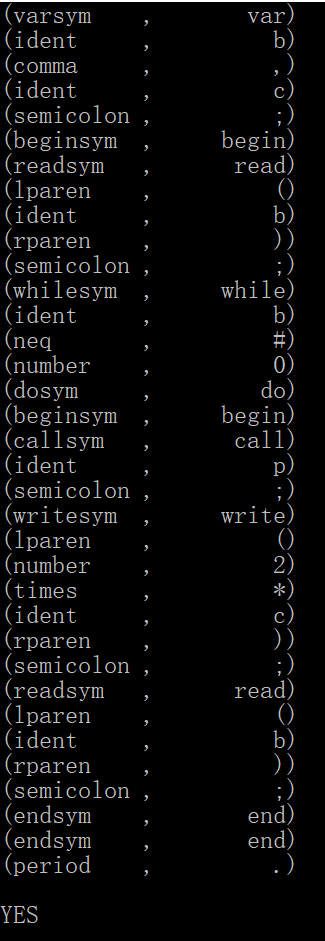
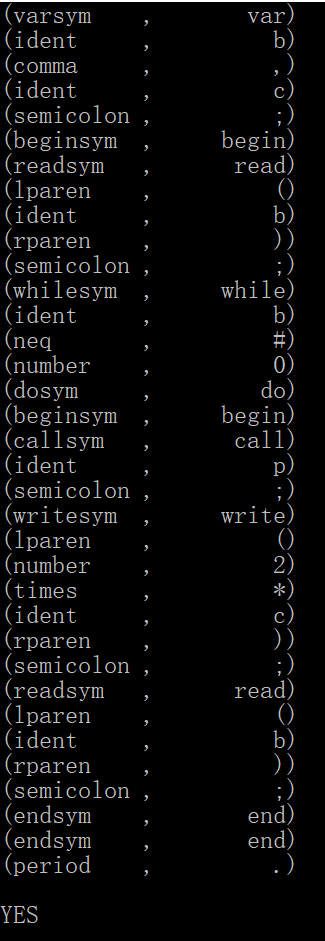
write(2 \* c);

read(b);

end

end.

**输出：**



**4、测试样例4：**

**输入：**

var m, n, r, q;

procedure gcd;

begin

while r#0 do

begin

q := m / n;

r := m - q \* n;

m := n;

n := r;

end

end;

begin

read(m) read(n);

if m < n

begin

r := m;

m := n;

n := r;

end;

begin

r:=1;

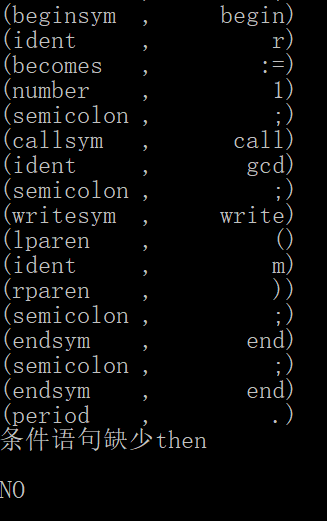
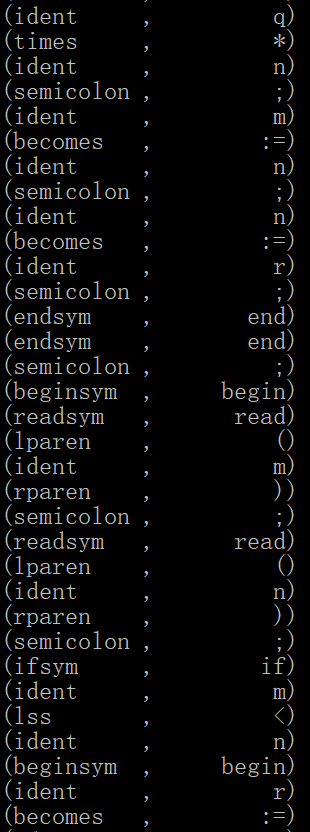
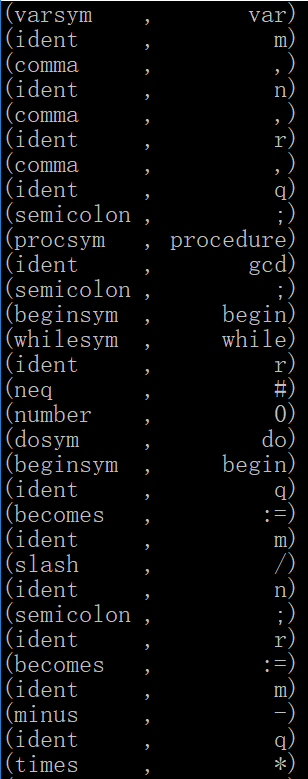
call gcd;

write(m);

end;

end.

**输出：**



**修改后：**

var m, n, r, q;

procedure gcd;

begin

while r#0 do

begin

q := m / n;

r := m -q \* n;

m := n;

n := r;

end

end;

begin

read(m); read(n);

if m < n then

begin

r := m;

m := n;

n := r;

end;

begin

r:=1;

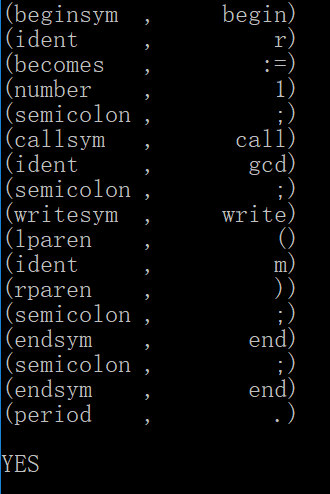
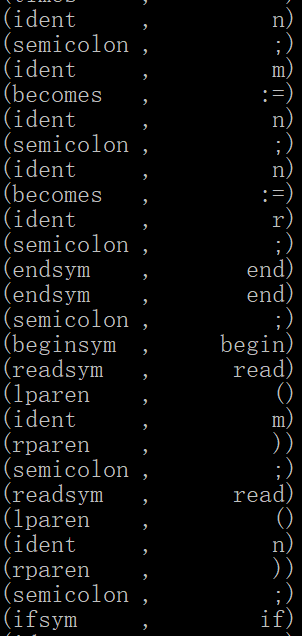
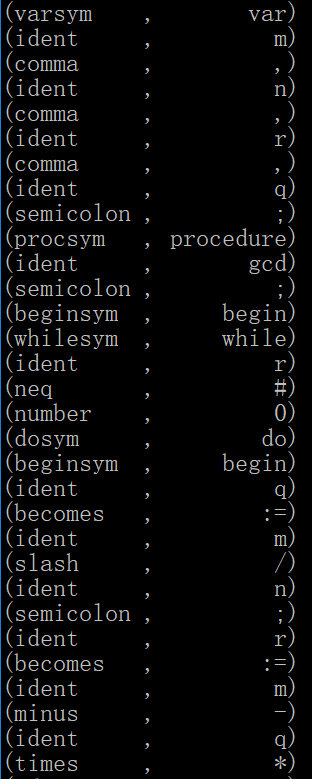
call gcd;

write(m);

end;

end.

**输出：**



**5、测试样例5：**

**输入：**

const

a = 10;

var

b,c;

procedure p;

var

d;

begin

d := 20;

c := d / a;

c := c + b;

if a < c then

c := 2 \* c

else

c := 2 \* c + 1;

begin

read(b);

while b<>0 do

begin

call p;

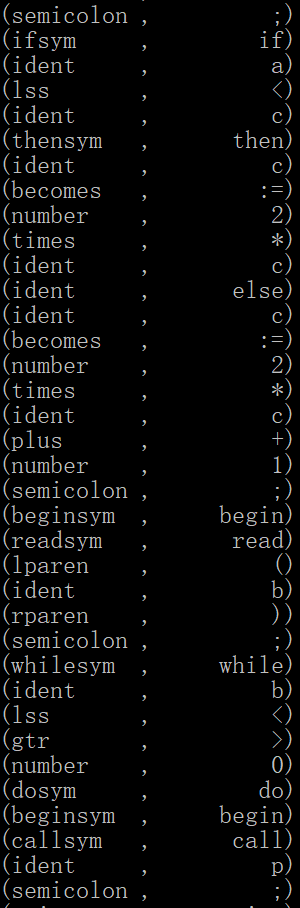
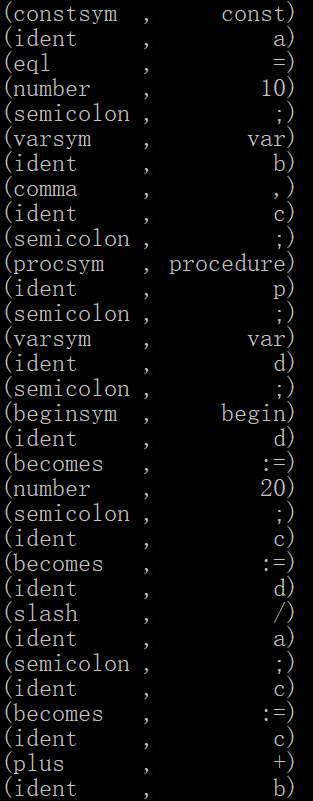
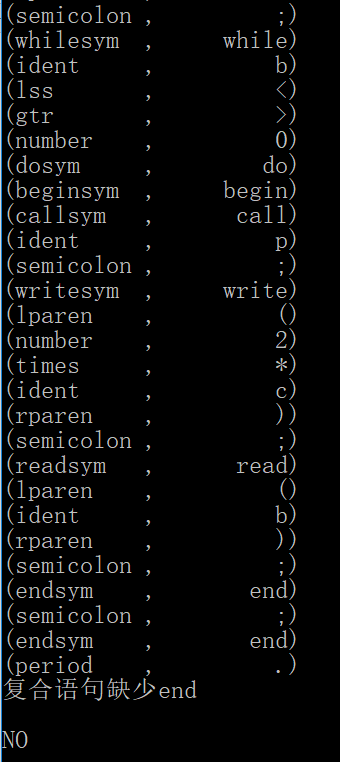
write(2 \* c);

read(b);

end;

end.

**输出：**

**修改后：**

const

a = 10;

var

b,c;

procedure p;

var

d;

begin

d := 20;

c := d / a;

c := c + b;

if a < c then

c := 2 \* c

end;

begin

read(b);

while b#0 do

begin

call p;

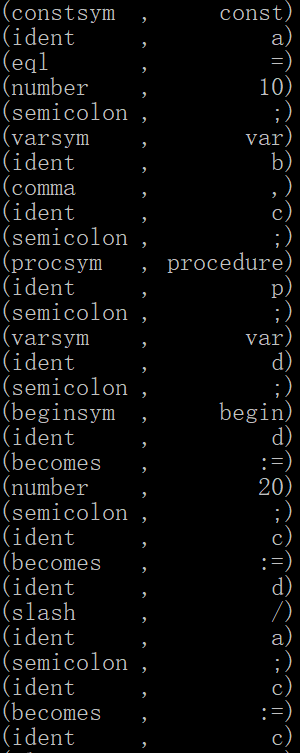
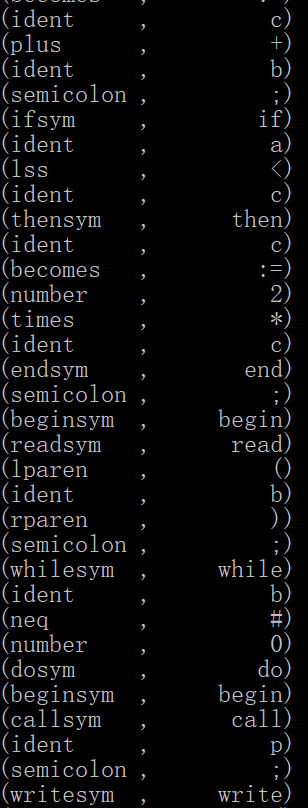
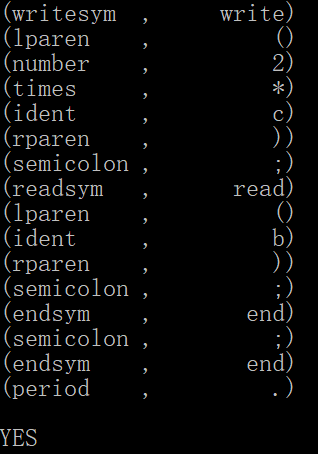
write(2 \* c);

read(b);

end;

end.

**输出：**

## 实验体会：

本次实验的主要目的是求编写 PL/0 语言的语法分析程序，实现对词法分析程序所提供的单词序列进行语法检查和结构分析，并且能够报出错误。我们小组对算术表达式、条件表达式、各类语句等PL/0语言文法进行了检查，在这个过程中遇到了许多问题，但经过小组讨论和查阅资料后得到了解决。我们是采用递归的方法去实现本次实验的，其实还有很多别的方法通过这次实验，都可以做进一步的尝试。这次实验让我进一步掌握常用的语法分析方法，而且也学会如何将书本中的知识灵活的运用到编程中，受益匪浅。