山东大学
SHANDONG UNIVERSITY

编译原理实验报告

PL/0 语言编译器实现

学院：泰山学堂

专业：计算机取向

学号：201605130116

姓名：杜洪超

目录

一 文法表示

PI/0 语言文法的 BNF 表示：

〈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(<表达式>{, <表达式>})
<字母> → a|b|c…x|y|z
<数字> → 0|1|2…7|8|9

二 词法分析

1　单词定义

由文法提取关键字，定义语言中所有的单词类别

```
/******************************** defind keywords ********************************/
const int KWNUM=15;
const string KEYWORD[] = {"IDENT","NUMBER","CONST","VAR","PROCEDURE","BEGIN","END","ODD","IF","THEN",
                          "CALL","WHILE","DO","READ","WRITE"};
const string symble[] = {"+","-","*","/","#","<","<=",">",">=","=",":=",",",".",";","(",")"};
/*********************************************************************************/
```

其中类型号 0 代表标识符，类型号 1 代表常数，2-15 类型号为一词一类的保留字

2 单词提取

2.1 数据格式

提取出的单词格式如下：

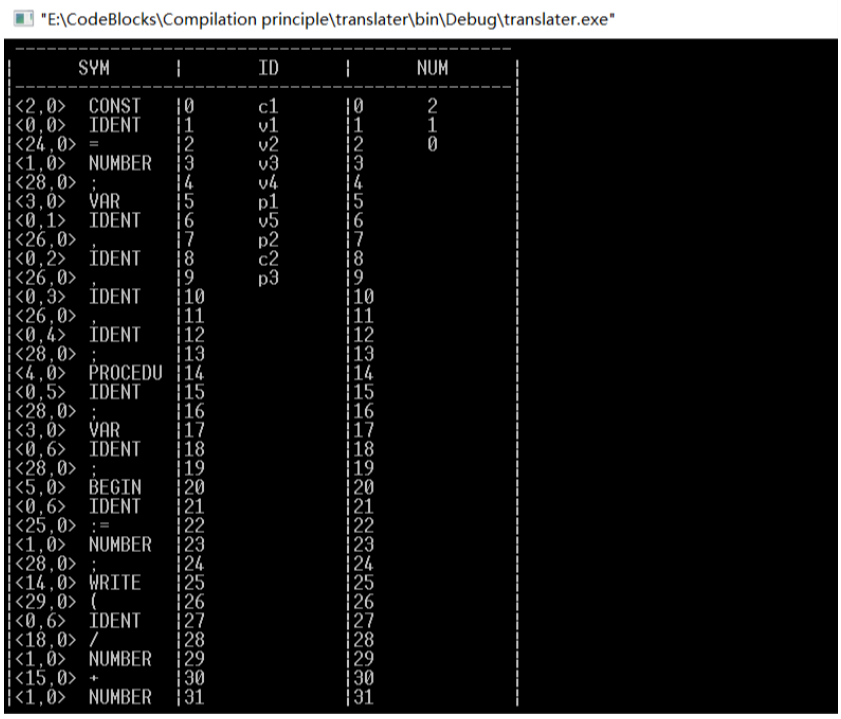
```
vector<pair<int,int> > SYM;
vector<string> ID;
vector<string> NUM;
```

所有单词都存在 SYM 数组中，其中每个单词第一项代表单词类别,对于非一词一类的单词，第二项代表其在相应表中的位置；

标识符和常数各使用一个表存放，其下标对应 SYM 数组中的第二项。

2.2 输出

构造 DFA 识别单词，读入每一个字符，将识别出的单词放在定义好的数据结构中，输出结果如下图：



上图对应的代码片段如下图：

```
const c1=2;
var v1,v2,v3,v4;
procedure p1;
var v5;
begin
     v5:=2;
     write(v5/2+2-1);
     while v3#0 do
        begin
            v4:=v1/v2;
            v3:=v1-v4*v2;
            v1:=v2;
            v2:=v3;
        end;
      end;
```

3 数据保存

词法分析从源文件读取数据，把提取的单词数据写入文件，供

语法分析使用，内容为定义的三组数据。

```
204 10 3
2 0
0 0
24 0
1 0
28 0
3 0
0 1
26 0
0 2
26 0
0 3
26 0
0 4
28 0
4 0
0 5
28 0
```

三 语法分析和目标代码生成

1 文法分析与修改

1.1 文法修改

修改原文法使其更符合递归下降的规则，主要对文法中[]以及{}处理，方法如下：

PI/0 语言文法的 BNF 表示：

〈程序〉→ <BLOCK>.

<BLOCK〉→〈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>};

+++ <常量说明部分> → CONST<多常量定义>; | ε

+++ <多常量定义> → <常量定义><任意常量定义>

+++ <任意常量定义> → , <常量定义><任意常量定义> | ε

<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>}; | ε

<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>} | ε

1.2 构造递归下降程序

根据修改后的产生式，构造递归下降函数如下：

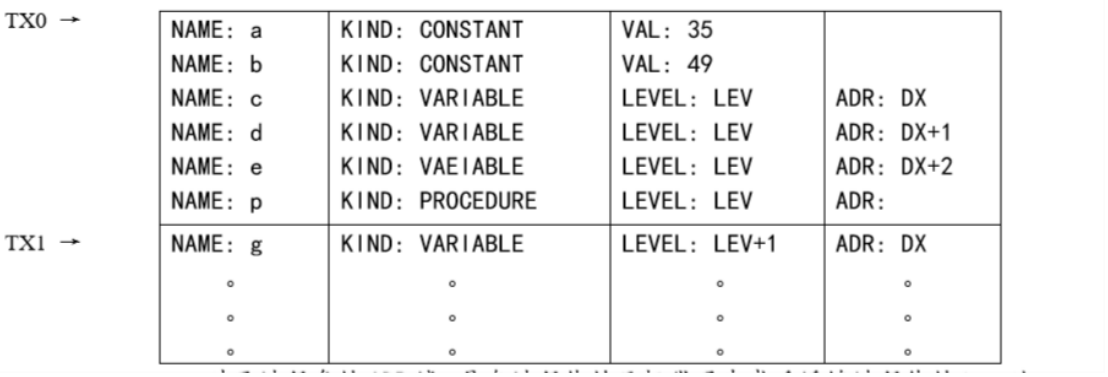
```
void BLOCK(int tx,int l);
void PROGRAM();
void SUBPROGRAM();
void CONST_DESCRIPTION();
void CONST_DEFINEDS();
void CONST_DEFINED_ANY();
void CONST_DEFINED();
void UNSIGNED_INT();
void VAR_DESCRIPTION();
void VAR_DEFINEDS();
void IDENTIFIER_ANY();
void IDENTIFIER();
void PROCEDURE_DESCRITION();
void PROCEDURES();
void PROCEDURE_ANY();
void PROCEDURE();
void PROCEDURE_HEAD();
void STATEMENT();
```

```
void ASSIGN();
void COMPLEX();
void STATEMENTS();
void STATEMENT_ANY();
void CONDITION();
void EXPRESSION();
void EXPRESSIONS();
void EXPRESSION_ANY();
void ITEM();
void ITEMS();
void ITEM_ANY();
void FACTOR_ANY();
void FACTOR();
void CONDITIONAL();
void CALL();
void WHILE();
void READ();
void WRITE();
```

2 说明部分处理

2.1 表格格式

对包括 main 函数在内的每个过程的说明部分，将相应信息填入表格，格式如下：



TX0 →	NAME: a	KIND: CONSTANT	VAL: 35	
	NAME: b	KIND: CONSTANT	VAL: 49	
	NAME: c	KIND: VARIABLE	LEVEL: LEV	ADR: DX
	NAME: d	KIND: VARIABLE	LEVEL: LEV	ADR: DX+1
	NAME: e	KIND: VAEIABLE	LEVEL: LEV	ADR: DX+2
	NAME: p	KIND: PROCEDURE	LEVEL: LEV	ADR:
TX1 →	NAME: g	KIND: VARIABLE	LEVEL: LEV+1	ADR: DX
	∘	∘	∘	∘
	∘	∘	∘	∘
	∘	∘	∘	∘

2.2 表格实现

表格通过两个一维数组分别实现表格主体 table 和指针数组 table_TX，其中指针数组类似与编译中 display 表的功能，供生成目标代码查找变量使用

```
#define MAXN 1000
typedef struct DESCRITION
{
        string name;
        string kind;
        union
        {
            int val;
            int level;
        };
        int adr;
} Descrition;
Descrition table[MAXN];
int table_index=0;
int TX[50];
int TX_index=0;
int level,dx;
```

2.3 表格生成

最终生成的表格如下图：

```
1   const c1=2;
2   var v1,v2,v3,v4;
3   procedure p1;
4    var v5;
5   begin
16  procedure p2;
17   const c2=2;
18   procedure p3;
19   begin
27  begin
34  begin
```

```
"E:\CodeBlocks\Compilation principle\Gramma analysis\bin\Debug\Gram...
0       c1      CONSTANT        2
1       v1      VARIABLE        0       3
2       v2      VARIABLE        0       4
3       v3      VARIABLE        0       5
4       v4      VARIABLE        0       6
5       p1      PROCEDURE       0       1
6       v5      VARIABLE        1       3
7       p2      PROCEDURE       0       47
8       c2      CONSTANT        2
9       p3      PROCEDURE       1       32
```

3 语句处理和目标代码生成

3.1 指令格式

指令格式如下：

PL/0 语言的目标指令是一种假想的栈式计算机的汇编语言，其格式如下：

f	l	a

其中 f 代表功能码，l 代表层次差，a 代表位移量。
目标指令有 8 条：
① LIT：将常数放到运栈顶，a 域为常数。
② LOD：将变量放到栈顶。a 域为变量在所说明层中的相对位置，l 为调用层与说明层的层差值。
③ STO：将栈顶的内容送到某变量单元中。a,l 域的含义与 LOD 的相同。
④ CAL：调用过程的指令。a 为被调用过程的目标程序的入中地址，l 为层差。
⑤ INT：为被调用的过程（或主程序）在运行栈中开辟数据区。a 域为开辟的个数。
⑥ JMP：无条件转移指令，a 为转向地址。
⑦ JPC：条件转移指令，当栈顶的布尔值为非真时，转向 a 域的地址，否则顺序执行。
⑧ OPR：关系和算术运算。具体操作由 a 域给出。运算对象为栈顶和次顶的内容进行运算，结果存放在次顶。a 域为 0 时是退出数据区。

```
typedef struct CODE
{
    string f;
    int l;
    int a;
} Code;
vector<Code> code;

/********************************** OPR **********************************/
string opra[]={"ret","+","-","*","/","#","<","<=",">",">=","=","%","read","write"};
               /*0   1   2   3   4   5   6    7   8    9   10  11   12      13*/
/********************************** OPR **********************************/
```

3.2 标识符查找

利用表格管理中的名字信息和索引数组，可以以静态链的方式找到变量而不会误判；

其中参数 name 为所找变量的名字，返回层差和偏移量放在 tl 和 tdx 中。

```
void lookup(string name,int* tl,int* tdx)
{
    bool findit=false;*tl=0;*tdx=3;
    for (int i=TX_index;i>=0;i--)
    {
        for (int j=TX[i];j<=table_index;j++)
        {
            if (table[j].kind=="PROCEDURE") break;
            if (table[j].name==name)
            {
                if (findit)ERROR();
                findit=true;
                if (table[j].kind=="CONSTANT")
                {
                    *tl=table[j].val;
                    *tdx=0;
                }
                if (table[j].kind=="VARIABLE")
                    *tdx=table[j].adr;
            }
         }
         if (findit)break;
         (*tl)++;
    }
    if (!findit) ERROR();
}
```

3.3 表达式计算

表达式使用递归下降函数规定优先级，对于同级的算符，使用符号栈把算符保存，按照文法在正确的时刻弹出计算。

3.4 过程入口回填

符号表中过程需要填写返回地址以供调用时跳转，但过程的首条指令并不能立刻产生，因此使用过程名栈，把所有的未填写入口地址的过程入栈，每当一个过程体结束，就弹出一个过程，回填入口地址。

3.5 过程调用

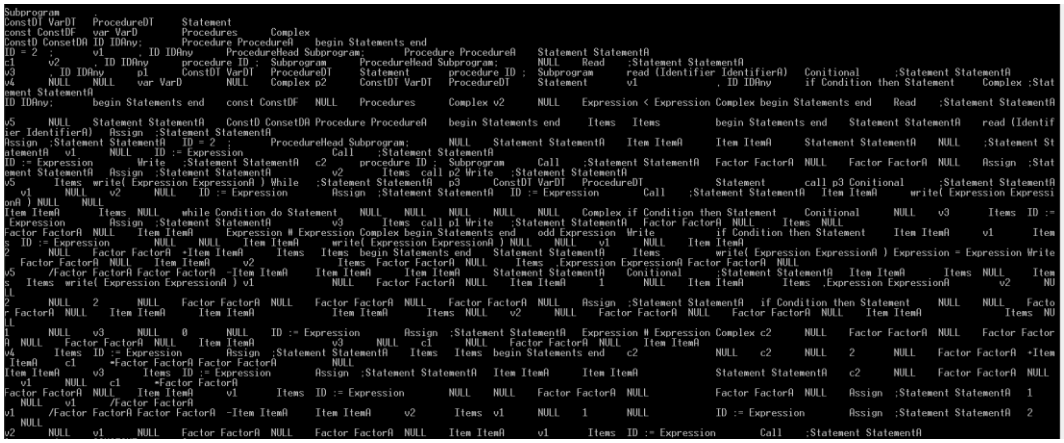
对于过程调用，要查找对应过程名的入口地址，过程调用要符合语言的限制，规定只允许调用子过程或兄弟过程，在符号表中找到对应的声明部分进行查找，返回层差和入口地址。

```
bool findit=false;
int l=0,a=0;
for (int i=TX[TX_index];i<table_index;i++)
    if(table[i].name==SYM_WORD&&table[i].kind=="PROCEDURE"&&table[i].level==level)
    {
        if (findit) {ERROR();break;}
        findit=true;
        l=level-table[i].level;
        a=table[i].adr;
    }
for (int i=TX[TX_index-1];i<TX[TX_index];i++)
    if(table[i].name==SYM_WORD&&table[i].kind=="PROCEDURE"&&table[i].level==(level-1))
    {
        if (findit) {ERROR();break;}
        findit=true;
        l=level-table[i].level;
        a=table[i].adr;
    }
if (!findit) {ERROR();return;}
```

4 输出

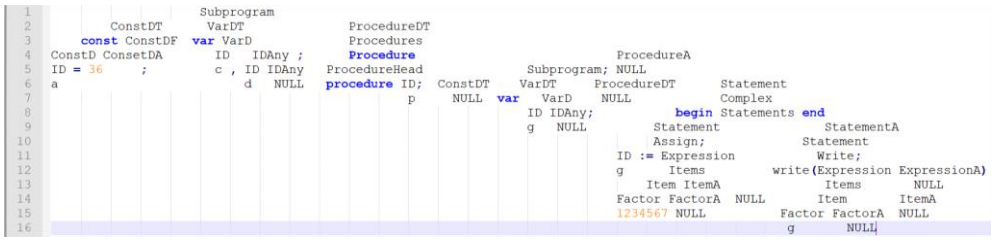
4.1 控制台输出

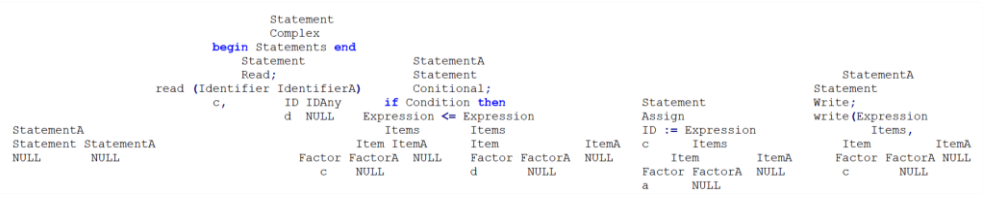
通过把子节点放到父节点下一行的方式，在命令行里输出语法树，结果如下：

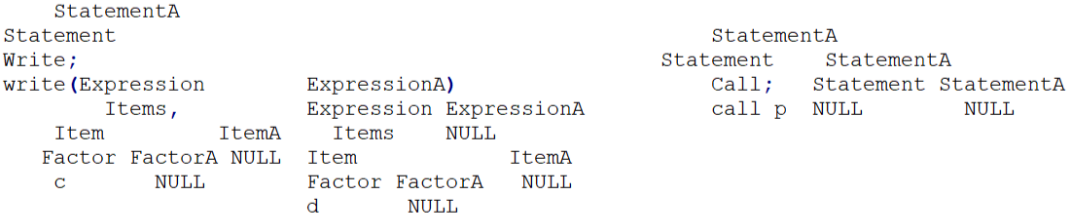


4.2 控制台输出(调整)

控制台输出结构十分混乱，将其复制到文本文档中，调整单词间隔后效果如下：

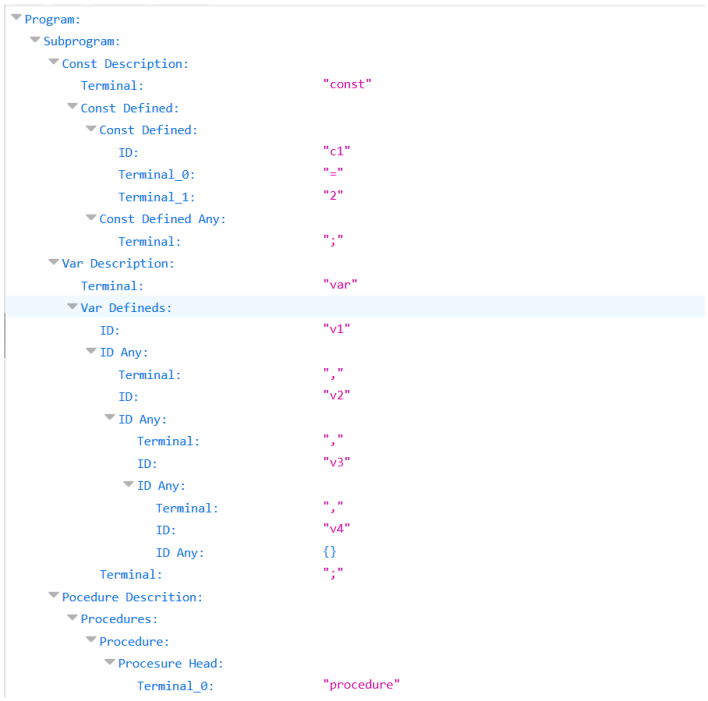






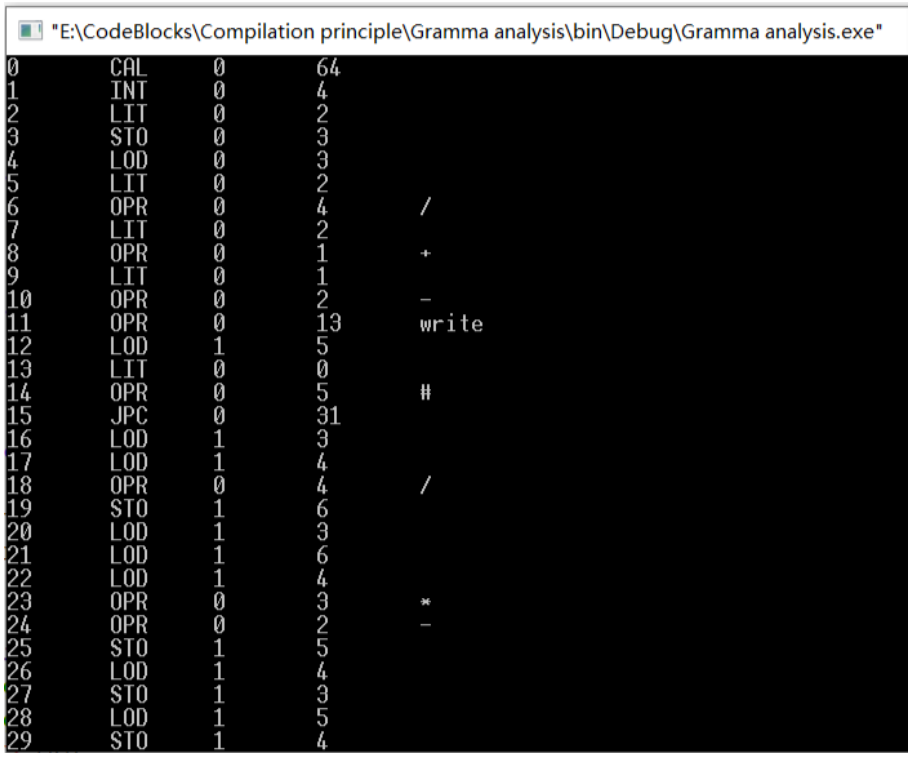
4.3 JSON 输出

横向扩展的语法树同样不直观，因此使用 json 格式重新构造语法树，使用 json 可视化工具打开 json 文件，可以实现较好的展示效果：



4.4 数据保存

语法分析的输出为目标代码，供解释执行程序读取；




```
CAL     0   64
INT     0   4
LIT     0   2
STO     0   3
LOD     0   3
LIT     0   2
OPR     0   4
LIT     0   2
OPR     0   1
LIT     0   1
OPR     0   2
OPR     0   13
LOD     1   5
LIT     0   0
OPR     0   5
JPC     0   31
LOD     1   3
LOD     1   4
OPR     0   4
STO     1   6
LOD     1   3
LOD     1   6
LOD     1   4
OPR     0   3
```

四 解释执行

1 数据定义

解释执行时的数据区定义如下所示：

为解释程序定义四个寄存器：

1. I：指令寄存器，存放当前正在解释的一条目标指令。
2. P：程序地址寄存器，指向下一条要执行的目标指令（相当于 CODE 数组的下标）。
3. T：栈顶寄存器，每个过程运行时要为它分配数据区（或称为数据段），该数据区分为两部分。
   静态部分：包括变量存放区和三个联单元。
   动态部分：作为临时工作单元和累加器用。需要时临时分配，用完立即释放。栈顶寄存器 T 指出了当前栈中最新分配的单元（T 也是数组 S 的下标）。
4. B：基地址寄存器，指出每个过程被调用时，在数据区 S 中给出它分配的数据段起始地址，也称为基地址。每个过程被调用时，在栈顶分配三个联系单元。这三个单元的内容分别是：
   SL：静态链，它是指向定义该过程的直接外过程运行时数据段的基地址。
   DL：动态链，它是指向调用该过程前正在运行过程的数据段的基地址。
   RA：返回地址，记录调用该过程时目标程序的断点，即当时的程序地址寄存器 P 的值。

2 指令执行

只介绍三类典型的指令如何执行，具体指令见源码；

2.1 LOD 指令

LOD 指令首先通过静态链找到变量的地址，再进行赋值操作；

```
if (I.f=="LOD")
{
    int base=B;
    for (int i=0;i<I.l;i++)
        base=Stack[base];
    if (Stack.size()>T)
        Stack[T]=Stack[base+I.a];
    else Stack.push_back(Stack[base+I.a]);
    T++;
}else
```

2.2 CAL 指令

CAL 指令完成的功能包括 CAL 和 INT 两条指令的功能，包括为新过程开辟空间，填写联系单元等；

```
if (I.f=="CAL")
{
    int ret=P;
    int level=I.l;
    P=I.a;
    I=code[P];
    P++;
    for (int i=0;i<I.a;i++)
    {
        if (Stack.size()>T) Stack[T]=0;
        else Stack.push_back(0);
        T++;
    }
    if (level==0)Stack[T-I.a]=B;
    else if (level==1)Stack[T-I.a]=Stack[B];
    Stack[T-I.a+1]=B;
    Stack[T-I.a+2]=ret;
    B=T-I.a;
}else
```

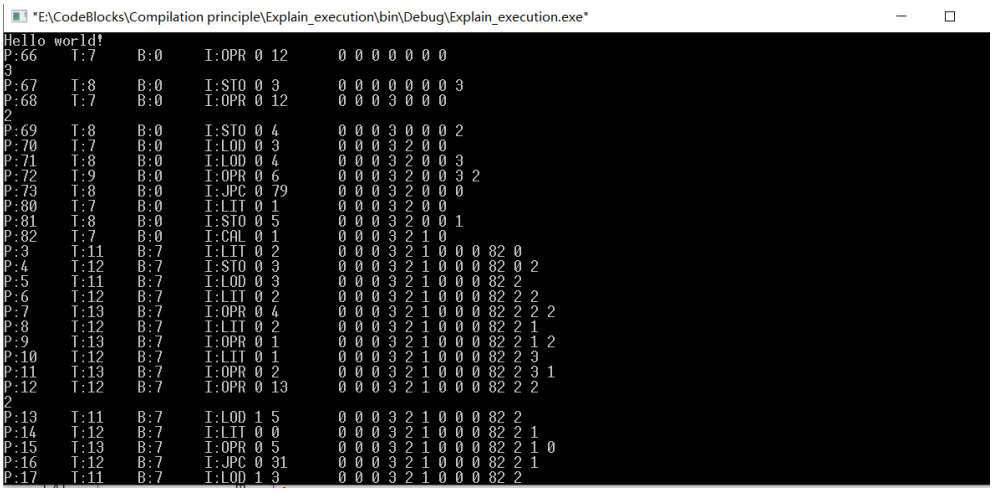
2.3 OPR 指令

OPR 指令根据具体的操作码执行相应的操作；

```
if (I.f=="OPR")
{
    int base;
    switch (I.a)
    {
    case 0:
        P=Stack[B+2];
        base=B;
        B=Stack[B+1];
        T=base;
        break;
    case 1:
        T-=1;
        Stack[T-1]=Stack[T-1]+Stack[T];
        break;
    case 2:
        T-=1;
        Stack[T-1]=Stack[T-1]-Stack[T];
        break;
```

3 输出

解释执行输出如下，包括寄存器和栈以及标准输出三部分：



五 源代码

1 词法分析

```
#include <iostream>
#include <map>
#include <vector>
#include <fstream>

using namespace std;

/********************** defind keywords *************************/
const int KWNUM=15;
const string KEYWORD[] =
{"IDENT","NUMBER","CONST","VAR","PROCEDURE","BEGIN","END","ODD","IF",
"THEN",
                      "CALL","WHILE","DO","READ","WRITE"};
const string symble[] =
{"+","-","*","/","#","<","<=",">",">=","=",":=",",",".",";","(",")"};
/****************************************************************/

map<string,int> KWTABLE; //map to find the kind of word

vector<pair<int,int> > SYM;
vector<string> ID;
vector<string> NUM;
fstream file("test.txt");     //source code
ofstream result("word1.txt"); //output word file

// look up the map for kind
int Reserve(string tem){if (KWTABLE.find(tem)!=KWTABLE.end())return
KWTABLE.find(tem)->second;else return 0;}
void GETSYM();

int main()
{
    for (int i = 0;i < KWNUM;i++) KWTABLE.insert(make_pair(KEYWORD[i],i));
    //make the map
    GETSYM(); //main work
    /****************** display on console ***********************/
    cout<<" ---------------------------------------------- "<<endl;
    cout<<"|      SYM      |      ID      |      NUM      |"<<endl;
    cout<<"|----------------------------------------------|"<<endl;
    for (int i=0;i<SYM.size();i++)
    {
        cout <<"|<" << SYM[i].first << "," << SYM[i].second << ">\t";
        if (SYM[i].first<15)
            cout << KEYWORD[SYM[i].first].substr(0,7) <<"\t|";
```

```
        else
            cout << symble[SYM[i].first-15]<<"\t|";
        if (i<ID.size())
            cout<< i << "\t" << ID[i].substr(0,7) << "\t|";
        else cout<<i<<"\t\t|";
        if (i<NUM.size())
            cout<< i << "\t" << NUM[i] << "\t|"<< endl;
        else cout<<i<<"\t\t|"<<endl;
    }
    cout<<"-------------------------------------------------"<<endl;

/*****************************************************************/

/************************  write to file ************************/
    result<<SYM.size()<<" "<<ID.size()<<" "<<NUM.size()<<endl;
    for (int i=0;i<SYM.size();i++)
        result<< SYM[i].first << " " << SYM[i].second<<endl;
    for (int i=0;i<ID.size();i++)
        result<< ID[i] <<endl;
    for (int i=0;i<NUM.size();i++)
        result<< NUM[i] <<endl;

/*****************************************************************/
    return 0;
}

void GETSYM()
{
    char c;
    int code,value;
    string strToken;
    while (!file.eof())
    {
        strToken = "";
        file.get(c);
        while (c == ' ' || c == '\n' || c == '\t')
        {
            file.get(c);
            if (file.eof()) return;
        }
        if (file.eof()) return;
        /*
        GetChar();
        GetBC();
```

```
        */
        if (isalpha(c)) //isLetter()
        {
            while(isalnum(c)) //isLetter() or isDigit()
            {
                strToken.push_back(c); //Concat();
                file.get(c);          //GetChar();
                if (file.eof()) break;
                //Retract();
            }
            if (!file.eof())file.unget();
            /**************** Case conversion ******************/
            string tem="";
            for (int i=0;i<strToken.length();i++)
                if (strToken[i]<=122&&strToken[i]>=97)
tem+=strToken[i]-32;
                    else tem+=strToken[i];
            /************************************************/
            code = Reserve(tem);
            if (code != 0)
                SYM.push_back(make_pair(code,0));  //keyword
            else
            {
                if (strToken.length()>10)
                {
                    cout << "The length of the identifier exceeds ten." <<
endl;
                    return;
                }
                /************* Filter duplicate words ************/
                bool f=false;
                for (int i=0;i<ID.size();i++)
                    if (ID[i]==strToken){f=true;value=i;}
                if (!f)
                {
                    value = ID.size();
                    ID.push_back(strToken);
                }
                /************************************************
                value = ID.size();
                ID.push_back(strToken);
                /************************************************/
                SYM.push_back(make_pair(0,value));
            }
```

```
}else if(isdigit(c))
{
    while(isdigit(c)) //isLetter() or isDigit()
    {
        strToken.push_back(c); //Concat();
        file.get(c);           //GetChar();
        if (file.eof()) break;
        //Retract();
    }
    if (!file.eof())file.unget();
    /*************** Filter duplicate numbers ***************/
    bool f=false;
    for (int i=0;i<NUM.size();i++)
        if (NUM[i]==strToken){ f=true;value=i;}
    if (!f)
    {
        value = NUM.size();
        NUM.push_back(strToken);
    }
    /*********************************************************
    value = NUM.size();
    NUM.push_back(strToken);
    *********************************************************/
    SYM.push_back(make_pair(1,value));
}else if (c == '+')
    SYM.push_back(make_pair(15,0));
else if (c == '-')
    SYM.push_back(make_pair(16,0));
else if (c == '*')
    SYM.push_back(make_pair(17,0));
else if (c == '/')
    SYM.push_back(make_pair(18,0));
else if (c == '#')
    SYM.push_back(make_pair(19,0));
else if (c == '<' || c == '>')
{
    char cc = file.get();
    if (cc == '=')
    {
        if (c == '<') SYM.push_back(make_pair(21,0));
        if (c == '>') SYM.push_back(make_pair(23,0));
    }
    else
    {
```

```
            if (c == '<') SYM.push_back(make_pair(20,0));
            if (c == '>') SYM.push_back(make_pair(22,0));
            file.unget();
        }
    }else if (c == '=')
        SYM.push_back(make_pair(24,0));
    else if (c == ':')
    {
        char cc = file.get();
        if (cc == '=') SYM.push_back(make_pair(25,0));
        else return;
    }else if (c == ',')
        SYM.push_back(make_pair(26,0));
    else if (c == '.')
        SYM.push_back(make_pair(27,0));
    else if (c == ';')
        SYM.push_back(make_pair(28,0));
    else if (c == '(')
        SYM.push_back(make_pair(29,0));
    else if (c == ')')
        SYM.push_back(make_pair(30,0));
    else return;
  }
}
```

2　语法分析和目标代码生成

```
#include <iostream>
#include <fstream>
#include <vector>
#include <stack>
#include <utility>

#define MAXN 1000

using namespace std;

void BLOCK(int tx,int l);
void PROGRAM();
void SUBPROGRAM();
void CONST_DESCRIPTION();
void CONST_DEFINEDS();
void CONST_DEFINED_ANY();
void CONST_DEFINED();
```

```
void UNSIGNED_INT();
void VAR_DESCRIPTION();
void VAR_DEFINEDS();
void IDENTIFIER_ANY();
void IDENTIFIER();
void PROCEDURE_DESCRITION();
void PROCEDURES();
void PROCEDURE_ANY();
void PROCEDURE();
void PROCEDURE_HEAD();
void STATEMENT();
void ASSIGN();
void COMPLEX();
void STATEMENTS();
void STATEMENT_ANY();
void CONDITION();
void EXPRESSION();
void EXPRESSIONS();
void EXPRESSION_ANY();
void ITEM();
void ITEMS();
void ITEM_ANY();
void FACTOR_ANY();
void FACTOR();
void CONDITIONAL();
void CALL();
void WHILE();
void READ();
void WRITE();

typedef struct DESCRITION
{
    string name;
    string kind;
    union
    {
        int val;
        int level;
    };
    int adr;
} Descrition;

typedef struct CODE
{
```

```
    string f;
    int l;
    int a;
} Code;
vector<Code> code;

const string KEYWORD[] =
{"IDENT","NUMBER","CONST","VAR","PROCEDURE","BEGIN","END","ODD","IF",
"THEN","CALL","WHILE","DO","READ","WRITE"};
const string symble[] =
{"+","-","*","/","#","<","<=",">",">=","=",":=",",",".",";","(",")"};
/**** OPR ****/
string
opra[]={"ret","+","-","*","/","#","<","<=",">",">=","=","%","read","w
rite"}; /*0    1   2    3   4    5    6    7    8    9    10   11   12
13*/
/**** OPR ****/

/************* Input and output ***************/
fstream word("word1.txt");
fstream cf("code.txt");
fstream Tree("Tree.json");
string output[1000];
int output_index=1;
string tree="";
/*******************************************/

vector<pair<int,int> > SYM;
vector<string> ID;
vector<string> NUM;
stack<string> opr;  //operator stack

int index=0;
string SYM_WORD;
Descrition table[MAXN];
int table_index=0;
int TX[50];
int TX_index=0;
int level,dx;

bool vard=false; //var_defind
bool is_read=false;
bool is_write=false;
stack<int> pro_index; //procedure index
```

```
int first=0; //Entry address

bool Error=false;

//output for console
void write_output(string s)
{
    output_index--;
    output[output_index]+=s;
    output_index++;
}

void ERROR()
{
    Error=true;
    cout<<"ERROR"<<endl;
    output[output_index]+="ERROR ";
}

void ADVANCE();
void init();
bool is_relational(string s);
bool is_identifier(string s){return SYM[index-1].first==0;}
bool is_unsignnum (string s){return SYM[index-1].first==1;}
void Output();

int main()
{
    init();
    PROGRAM();
    Output();
    return 0;
}

//Read words
void init()
{
    int m,n,x;
    word>>m;
    word>>n;
    word>>x;
    int first,second;
    for (int i=0; i<m; i++)
    {
```

```
        word>>first;
        word>>second;
        SYM.push_back(make_pair(first,second));
    }
    string tem;
    for (int i=0; i<n; i++)
    {
        word>>tem;
        ID.push_back(tem);
    }
    for (int i=0; i<x; i++)
    {
        word>>tem;
        NUM.push_back(tem);
    }
    ADVANCE();
    for (int i=0; i<1000; i++) output[i]="";
}

void Output()
{
    //tree in concole
    /*for (int i=0; i<25; i++)
        cout<<output[i]<<endl;*/
    //table
    for (int i=0; i<table_index; i++)
    {
        cout<<i<<"\t"<<table[i].name<<"\t"<<table[i].kind<<"\t";
        if (table[i].kind=="CONSTANT")
            cout<<table[i].val<<endl;
        else cout<<table[i].level<<"\t"<<table[i].adr<<endl;
    }
    //code
    for (int i=0; i<code.size(); i++)
    {
    if (code[i].f=="OPR")
cout<<i<<"\t"<<code[i].f<<"\t"<<code[i].l<<"\t"<<code[i].a<<"\t"<<opr
a[code[i].a]<<endl;
        else
cout<<i<<"\t"<<code[i].f<<"\t"<<code[i].l<<"\t"<<code[i].a<<endl;
        cf<<code[i].f<<"\t\t"<<code[i].l<<"\t"<<code[i].a<<endl;
    }
    cout<<"main "<<first<<endl;
    //json
```

```
    Tree<<tree<<endl;
}

int str2num(string s)
{
    int num=0;
    for (int i=0; i<s.length(); i++)
        num=num*10+s[i]-'0';
    return num;
}

void fill_table(string name,string kind,string value)
{
    if (kind=="CONSTANT")
    {
        table[table_index].kind=kind;
        table[table_index].name=name;
        table[table_index].val=str2num(value);
        table[table_index].adr=0;
    }
    if (kind=="VARIABLE")
    {
        table[table_index].kind=kind;
        table[table_index].name=name;
        table[table_index].level=level;
        table[table_index].adr=dx;
        dx++;
    }
    if (kind=="PROCEDURE")
    {
        table[table_index].kind=kind;
        table[table_index].name=name;
        table[table_index].level=level;
    }
    table_index++;
}

//look up table and return Level difference and offset
void lookup(string name,int* tl,int* tdx)
{
    bool findit=false;*tl=0;*tdx=3;
    for (int i=TX_index;i>=0;i--)
    {
        for (int j=TX[i];j<=table_index;j++)
```

```
        {
            if (table[j].kind=="PROCEDURE") break;
            if (table[j].name==name)
            {
                if (findit)ERROR();
                findit=true;
                if (table[j].kind=="CONSTANT")
                {
                    *tl=table[j].val;
                    *tdx=0;
                }
                if (table[j].kind=="VARIABLE")
                    *tdx=table[j].adr;
            }
         }
         if (findit)break;
         (*tl)++;
    }
    if (!findit) ERROR();
}

void emit(string s,int l,int a)
{
    cout<<s<<" "<<l<<" "<<a<<endl;
    Code t;
    t.f=s;t.l=l;t.a=a;
    code.push_back(t);
}

void putonstack(string s)
{
    if (is_identifier(s))
    {
        int tl=0;int tdx=3;
        lookup(s,&tl,&tdx);
        if(tdx==0)emit("LIT",0,tl);
        else emit("LOD",tl,tdx);
    }else
    if(is_unsignnum(s))
        emit("LIT",0,str2num(s));
}

void ADVANCE()
{
```

```
    if (SYM[index].first==0)
        SYM_WORD=ID[SYM[index].second];
    else if (SYM[index].first==1)
        SYM_WORD=NUM[SYM[index].second];
    else if (SYM[index].first<15)
        SYM_WORD=KEYWORD[SYM[index].first];
    else
        SYM_WORD=symble[SYM[index].first-15];
    for (int i=0; i<SYM_WORD.length(); i++)
        if (SYM_WORD[i]<=90&&SYM_WORD[i]>=65)
            SYM_WORD[i]+=32;
    index++;
    cout<<index<<" "<<SYM_WORD<<endl;
}

void BLOCK(int tx,int l)
{
    TX[TX_index]=tx;
    level=l;dx=3;
    output[output_index]+="Subprogram\t";
    tree+="\"Subprogram\":{";
    output_index++;
    SUBPROGRAM();
    tree+="}";
    output_index--;
}

void PROGRAM()
{
    emit("CAL",0,0);
    tree+="{\"Program\":{";
    BLOCK(0,0);
    if (SYM_WORD==".")
        {output[output_index]+=".\t";tree+=",\"Terminal\":\".\"";}
    else ERROR();
    tree+="}}";
    if (index<SYM.size())
        ERROR();
}

void SUBPROGRAM()
{
    if (Error)return;
    tree+="\"Const Description\":{";
```

```
    output[output_index]+="ConstDT\t";
    output_index++;
    CONST_DESCRIPTION();
    tree+="},";
    output_index--;
    output[output_index]+="VarDT\t";
    output_index++;
    tree+="\"Var Description\":{";
    VAR_DESCRIPTION();
    tree+="},";
    output_index--;
    output[output_index]+="ProcedureDT\t";
    output_index++;
    tree+="\"Pocedure Descrition\":{";
    PROCEDURE_DESCRITION();
    tree+="},";
    output_index--;
    output[output_index]+="Statement\t";
    output_index++;
    if (!pro_index.empty())
    {
        int i=pro_index.top();
        table[i].adr=code.size();
        pro_index.pop();
    }
    else
        {first=code.size();code[0].a=first;}
    int num=0;
    for (int i=TX[TX_index];i<table_index;i++)
    {
        if (table[i].kind=="VARIABLE") num++;
        if (table[i].kind=="PROCEDURE") break;
    }
    emit("INT",0,num+3);
    tree+="\"Statement\":{";
    STATEMENT();
    tree+="}";
    emit("OPR",0,0);
    output_index--;
}

void CONST_DESCRIPTION()
{
    if (Error)return;
```

```
    if (SYM_WORD=="const")
    {
        ADVANCE();
        output[output_index]+="const ConstDF\t";
        tree+="\"Terminal\":\"const\",\"Const Defined\":{";
        output_index++;
        CONST_DEFINEDS();
        tree+="}";
        output_index--;
    }
    else
        output[output_index]+="NULL\t";
}

void CONST_DEFINEDS()
{
    if (Error)return;
    output[output_index]+="ConstD ConsetDA\t";
    output_index++;
    tree+="\"Const Defined\":{";
    CONST_DEFINED();
    tree+="},\"Const Defined Any\":{";
    CONST_DEFINED_ANY();
    tree+="}";
    output_index--;
}

void CONST_DEFINED_ANY()
{
    if (Error)return;
    if (SYM_WORD==",")
    {
        ADVANCE();
        output[output_index]+=", ConstD ConsetDA\t";
        tree+="\"Terminal\":\",\"";
        output_index++;
        tree+="\"Const Defined\":{";
        CONST_DEFINED();
        tree+="},\"Const Defined Any\":{";
        CONST_DEFINED_ANY();
        tree+="}";
        output_index--;
    }
    else if (SYM_WORD==";")
```

```
    {
        ADVANCE();
        tree+="\"Terminal\":\";\"";
        output[output_index]+=";\t";
    }
    else ERROR();
}

void CONST_DEFINED()
{
    if (Error)return;
    output[output_index]+="ID ";
    output_index++;
    string name=SYM_WORD;
    tree+="\"ID\":";
    IDENTIFIER();
    tree+=",";
    output_index--;
    if (SYM_WORD=="=")
    {
        ADVANCE();
        output[output_index]+="= ";
        tree+="\"Terminal_0\":\"=\",";
        if (is_unsignnum(SYM_WORD))
        {
            fill_table(name,"CONSTANT",SYM_WORD);
            tree+="\"Terminal_1\":\"";
            tree+=SYM_WORD;
            tree+="\"";
            output[output_index]+=SYM_WORD;
            output[output_index]+="\t";
            ADVANCE();
        }
        else
            ERROR();
    }
    else
        ERROR;
}

void VAR_DESCRIPTION()
{
    if (Error)return;
    vard=true;
```

```
    if (SYM_WORD=="var")
    {
        output[output_index]+="var VarD\t";
        ADVANCE();
        output_index++;
        tree+="\"Terminal\":\"var\",\"Var Defineds\":{";
        VAR_DEFINEDS();
        tree+="}";
        output_index--;
    }
    else
        output[output_index]+="NULL\t";
    vard=false;
}

void VAR_DEFINEDS()
{
    if (Error)return;
    output[output_index]+="ID IDAny";
    output_index++;
    fill_table(SYM_WORD,"VARIABLE","");
    tree+="\"ID\":";
    IDENTIFIER();
    tree+=",\"ID Any\":{";
    IDENTIFIER_ANY();
    tree+="},";
    output_index--;
    if (SYM_WORD==";")
    {
        tree+="\"Terminal\":\";\"";
        ADVANCE();
        output[output_index]+=";\t";
    }
    else ERROR();
}

void IDENTIFIER_ANY()
{
    if (Error)return;
    if (SYM_WORD==",")
    {
        tree+="\"Terminal\":\",\",";
        output[output_index]+=", ID IDAny\t";
        ADVANCE();
```

```
        output_index++;
        if (vard)fill_table(SYM_WORD,"VARIABLE","");
        tree+="\"ID\":";
        IDENTIFIER();
        tree+=",\"ID Any\":{";
        IDENTIFIER_ANY();
        tree+="}";
        output_index--;
    }
    else
        output[output_index]+="NULL\t";
}

void IDENTIFIER()
{
    if (Error)return;
    if (is_identifier(SYM_WORD))
    {
        if (is_read)
        {
            emit("OPR",0,12);
            int l,a;
            lookup(SYM_WORD,&l,&a);
            emit("STO",l,a);
        }
        tree+="\"";
        tree+=SYM_WORD;
        tree+="\"";
        output[output_index]+=SYM_WORD;
        output[output_index]+="\t";
        ADVANCE();
    }
    else ERROR();
}

void PROCEDURE_DESCRITION()
{
    if (Error)return;
    if (SYM_WORD=="procedure")
    {
        output[output_index]+="Procedures\t";
        output_index++;
        tree+="\"Procedures\":{";
        PROCEDURES();
```

```
        tree+="}";
        output_index--;
    }
    else
        output[output_index]+="NULL\t";
}

void PROCEDURES()
{
    if (Error)return;
    output[output_index]+="Procedure ProcedureA\t";
    output_index++;
    tree+="\"Procedure\":{";
    PROCEDURE();
    tree+="},\"Procedure Any\":{";
    PROCEDURE_ANY();
    tree+="}";
    output_index--;
}

void PROCEDURE_ANY()
{
    if (Error)return;
    if (SYM_WORD=="procedure")
    {
        output[output_index]+="Procedure ProcedureA\t";
        output_index++;
        tree+="\"Procedure\":{";
        PROCEDURE();
        tree+="},\"Procesure Any\":{";
        PROCEDURE_ANY();
        tree+="}";
        output_index--;
    }
    else
        output[output_index]+="NULL\t";
}

void PROCEDURE()
{
    if (Error)return;
    output[output_index]+="ProcedureHead Subprogram";
    output_index++;
    pro_index.push(table_index);
```

```
    tree+="\"Procesure Head\":{";
    PROCEDURE_HEAD();
    tree+="},";
    TX_index++;
    level++;
    BLOCK(table_index,level);
    level--;
    TX_index--;
    output_index--;
    if (SYM_WORD==";")
    {
        output[output_index]+=";\t";
        tree+=",\"Terminal\":\";\"";
        ADVANCE();
    }
    else
        ERROR();
}

void PROCEDURE_HEAD()
{
    if (Error)return;
    if (SYM_WORD=="procedure")
    {
        ADVANCE();
        output[output_index]+="procedure ID ";
        output_index++;
        fill_table(SYM_WORD,"PROCEDURE","");
        tree+="\"Terminal_0\":\"procedure\",\"ID\":";
        IDENTIFIER();
        tree+=",";
        output_index--;
        if (SYM_WORD==";")
        {
            tree+="\"Terminal_1\":\";\"";
            output[output_index]+=";\t";
            ADVANCE();
        }
        else
            ERROR();
    }
    else
        ERROR();
}
```

```
void STATEMENT()
{
    if (Error)return;
    if (is_identifier(SYM_WORD))
    {
        output[output_index]+="Assign\t";
        output_index++;
        tree+="\"Assign\":{";
        ASSIGN();
        tree+="}";
        output_index--;
    }
    else if (SYM_WORD=="if")
    {
        output[output_index]+="Conitional\t";
        output_index++;
        tree+="\"Conditional\":{";
        CONDITIONAL();
        tree+="}";
        output_index--;
    }
    else if (SYM_WORD=="while")
    {
        output[output_index]+="While\t";
        output_index++;
        tree+="\"While\":{";
        WHILE();
        tree+="}";
        output_index--;
    }
    else if (SYM_WORD=="call")
    {
        output[output_index]+="Call\t";
        output_index++;
        tree+="\"Call\":{";
        CALL();
        tree+="}";
        output_index--;
    }
    else if (SYM_WORD=="write")
    {
        output[output_index]+="Write\t";
        output_index++;
```

```
        tree+="\"Write\":{";
        WRITE();
        tree+="}";
        output_index--;
    }
    else if (SYM_WORD=="read")
    {
        output[output_index]+="Read\t";
        output_index++;
        tree+="\"Read\":{";
        READ();
        tree+="}";
        output_index--;
    }
    else if (SYM_WORD=="begin")
    {
        output[output_index]+="Complex\t";
        output_index++;
        tree+="\"Complex\":{";
        COMPLEX();
        tree+="}";
        output_index--;
    }
    else output[output_index]+="NULL\t";
}

void ASSIGN()
{
    if (Error)return;
    output[output_index]+="ID ";
    output_index++;
    int tl;int tdx;
    lookup(SYM_WORD,&tl,&tdx);
    tree+="\"ID\":";
    IDENTIFIER();
    tree+=",";
    output_index--;
    if (SYM_WORD==":=")
    {
        ADVANCE();
        output[output_index]+=":= Expression\t";
        output_index++;
        tree+="\"Terminal\":\":=\",\"Expression\":{";
        EXPRESSION();
```

```
        tree+="}";
        emit("STO",tl,tdx);
        output_index--;
    }
    else ERROR();
}

void COMPLEX()
{
    if (Error)return;
    if (SYM_WORD=="begin")
    {
        tree+="\"Terminal_0\":\"begin\",";
        output[output_index]+="begin Statements ";
        output_index++;
        ADVANCE();
        tree+="\"Statement\":{";
        STATEMENTS();
        tree+="},";
        output_index--;
        if (SYM_WORD=="end")
        {
            tree+="\"Terminal_1\":\"end\"";
            ADVANCE();
            output[output_index]+="end\t";
        }
        else
            ERROR();
    }
    else ERROR();
}

void STATEMENTS()
{
    if (Error)return;
    output[output_index]+="Statement StatementA\t";
    output_index++;
    tree+="\"Statement\":{";
    STATEMENT();
    tree+="},\"Statement Any\":{";
    STATEMENT_ANY();
    tree+="}";
    output_index--;
}
```

```
void STATEMENT_ANY()
{
    if (Error)return;
    if (SYM_WORD==";")
    {
        tree+="\"Terminal\":\";\",";
        output[output_index]+=";Statement StatementA\t";
        output_index++;
        ADVANCE();
        tree+="\"Statement\":{";
        STATEMENT();
        tree+="},\"Statement Any\":{";
        STATEMENT_ANY();
        tree+="}";
        output_index--;
    }
    else output[output_index]+="NULL\t";

}

void CONDITION()
{
    if (Error)return;
    if (SYM_WORD=="odd")
    {
        tree+="\"Terminal\":\"odd\",";
        output[output_index]+="odd Expression\t";
        output_index++;
        ADVANCE();
        tree+="\"Expression\":{";
        EXPRESSION();
        tree+="}";
        emit("LIT",0,2);
        emit("OPR",0,11);
        output_index--;
    }
    else
    {
        output[output_index]+="Expression ";
        output_index++;
        tree+="\"Expression_0\":{";
        EXPRESSION();
        tree+="},";
```

```
        output_index--;
        if (is_relational(SYM_WORD))
        {
            tree+="\"Terminal\":\"";
            tree+=SYM_WORD;
            tree+="\",";
            output[output_index]+=SYM_WORD;
            output[output_index]+=" Expression\t";
            int rela=0;
            for(int i=0;i<12;i++)
                if (SYM_WORD==opra[i]){cout<<"debug\t"<<SYM_WORD<<"
"<<i<<endl; rela=i;}
            ADVANCE();
            output_index++;
            tree+="\"Expression_1\":{";
            EXPRESSION();
            tree+="}";
            emit("OPR",0,rela);
            output_index--;
        }
        else ERROR();
    }
}

void EXPRESSION()
{
    if (Error)return;
    if (SYM_WORD=="-")
    {
        opr.push(SYM_WORD);
        emit("LIT",0,0);
    }
    if (SYM_WORD=="+"||SYM_WORD=="-")
    {
        tree+="\"Terminal\":\"";
        tree+=SYM_WORD;
        tree+="\",";
        output[output_index]+=SYM_WORD;
        ADVANCE();
    }
    output[output_index]+=" Items \t";
    output_index++;
    tree+="\"Items\":{";
    ITEMS();
```

```
    tree+="}";
    output_index--;
    if (is_write)
        emit("OPR",0,13);
}

void EXPRESSIONS()
{
    if (Error)return;
    output[output_index]+="Expression ExpressionA\t";
    output_index++;
    tree+="\"Expression\":{";
    EXPRESSION();
    tree+="},\"Expression Any\":{";
    EXPRESSION_ANY();
    tree+="}";
    output_index--;
}

void EXPRESSION_ANY()
{
    if (Error)return;
    if (SYM_WORD==",")
    {
        ADVANCE();
        output[output_index]+=",Expression ExpressionA\t";
        output_index++;
        tree+="\"Terminal\":\",\",";
        tree+="\"Expression\":{";
        EXPRESSION();
        tree+="},\"Expression Any\":{";
        EXPRESSION_ANY();
        tree+="}";
        output_index--;
    }
    else output[output_index]+="NULL\t";
}

void ITEMS()
{
    if (Error)return;
    output[output_index]+="Item ItemA\t";
    output_index++;
    tree+="\"Item\":{";
```

```
    ITEM();
    tree+="},";
    if (!opr.empty())
    {
        if (opr.top()=="-")
        {
            emit("OPR",0,3);
            opr.pop();
        }
    }
    tree+="\"Item Any\":{";
    ITEM_ANY();
    tree+="}";
    output_index--;
}

void ITEM_ANY()
{
    if (Error)return;
    if (SYM_WORD=="+"||SYM_WORD=="-")
    {
        tree+="\"Terminal\":\"";
        tree+=SYM_WORD;
        tree+="\",";
        output[output_index]+=SYM_WORD;
        output[output_index]+="Item ItemA\t";
        opr.push(SYM_WORD);
        ADVANCE();
        output_index++;
        tree+="\"Item\":{";
        ITEM();
        tree+="},";
        if (opr.top()=="+")
            emit("OPR",0,1);
        else
        if (opr.top()=="-")
            emit("OPR",0,2);
        else cout<<"opr.top()"<<opr.top()<<endl;;
        opr.pop();
        tree+="\"Item Any\":{";
        ITEM_ANY();
        tree+="}";
        output_index--;
    }
```

```
    else output[output_index]+="NULL\t";
}

void ITEM()
{
    if (Error)return;
    output[output_index]+="Factor FactorA\t";
    output_index++;
    tree+="\"Factor\":{";
    FACTOR();
    tree+="},\"Factor Any\":{";
    FACTOR_ANY();
    tree+="}";
    output_index--;
}

void FACTOR_ANY()
{
    if (Error)return;
    if (SYM_WORD=="*"||SYM_WORD=="/")
    {
        opr.push(SYM_WORD);
        tree+="\"Terminal\":\"";
        tree+=SYM_WORD;
        tree+="\",";
        output[output_index]+=SYM_WORD;
        output[output_index]+="Factor FactorA\t";
        output_index++;
        tree+="\"Facotr\":{";
        ADVANCE();
        FACTOR();
        tree+="},\"Factor Any\":{";
        FACTOR_ANY();
        tree+="}";
        output_index--;
    }
    else output[output_index]+="NULL\t";
}

void FACTOR()
{
    if (Error)return;
    if (is_identifier(SYM_WORD)||is_unsignnum(SYM_WORD))
    {
```

```
    if (!opr.empty()&&(opr.top()=="*"||opr.top()=="/"))
    {
        putonstack(SYM_WORD);
        if (opr.top()=="*")
            emit("OPR",0,3);
        if (opr.top()=="/")
            emit("OPR",0,4);
        opr.pop();
    }
    else
        putonstack(SYM_WORD);
    tree+="\"Terminal\":\"";
    tree+=SYM_WORD;
    tree+="\"";
    output[output_index]+=SYM_WORD;
    output[output_index]+="\t";
    ADVANCE();
}
else if (SYM_WORD=="(")
{
    opr.push(SYM_WORD);
    tree+="\"Terminal_0\":\"";
    tree+=SYM_WORD;
    tree+="\",";
    output[output_index]+="(";
    output[output_index]+="Expression ";
    ADVANCE();
    output_index++;
    tree+="\"Expression\":{";
    EXPRESSION();
    tree+="},";
    output_index--;
    if (SYM_WORD==")")
    {
        opr.pop();
        tree+="\"Terminal_1\":\"";
        tree+=SYM_WORD;
        tree+="\"";
        output[output_index]+=")\t";
        ADVANCE();
    }
    else
        ERROR();
}
```

```
    else ERROR();
}

void CONDITIONAL()
{
    if (Error)return;
    if (SYM_WORD=="if")
    {
        output[output_index]+="if ";
        output[output_index]+="Condition ";
        output_index++;
        tree+="\"Terminal_0\":\"";
        tree+=SYM_WORD;
        tree+="\",";
        ADVANCE();
        tree+="\"Condition\":{";
        CONDITION();
        tree+="},";
        emit("JPC",0,0);
        int index=code.size()-1;
        output_index--;
        if (SYM_WORD=="then")
        {
            tree+="\"Terminal_1\":\"";
            tree+=SYM_WORD;
            tree+="\",";
            output[output_index]+="then Statement\t";
            ADVANCE();
            output_index++;
            tree+="\"Statement\":{";
            STATEMENT();
            tree+="}";
            code[index].a=code.size();
            output_index--;
        }
        else ERROR();
    }
    else ERROR();
}

void CALL()
{
    if (Error)return;
    if (SYM_WORD=="call")
```

```
    {
        tree+="\"Terminal_0\":\"";
        tree+=SYM_WORD;
        tree+="\",";
        output[output_index]+="call ";
        ADVANCE();
        if (is_identifier(SYM_WORD))
        {
            bool findit=false;
            int l=0,a=0;
            for (int i=TX[TX_index];i<table_index;i++)

if(table[i].name==SYM_WORD&&table[i].kind=="PROCEDURE"&&table[i].leve
l==level)
                {
                    if (findit) {ERROR();break;}
                    findit=true;
                    l=level-table[i].level;
                    a=table[i].adr;
                }
            for (int i=TX[TX_index-1];i<TX[TX_index];i++)

if(table[i].name==SYM_WORD&&table[i].kind=="PROCEDURE"&&table[i].leve
l==(level-1))
                {
                    if (findit) {ERROR();break;}
                    findit=true;
                    l=level-table[i].level;
                    a=table[i].adr;
                }
            if (!findit) {ERROR();return;}
            emit("CAL",l,a);
            tree+="\"Terminal_1\":\"";
            tree+=SYM_WORD;
            tree+="\"";
            output[output_index]+=SYM_WORD;
            output[output_index]+="\t";
            ADVANCE();
        }
        else
            ERROR();
    }
    else
        ERROR();
```

```
}

void WHILE()
{
    if (Error)return;
    if (SYM_WORD=="while")
    {
        output[output_index]+="while ";
        output[output_index]+="Condition ";
        output_index++;
        tree+="\"Terminal_0\":\"";
        tree+=SYM_WORD;
        tree+="\",";
        ADVANCE();
        int index0=code.size();
        tree+="\"Condition\":{";
        CONDITION();
        tree+="},";
        int index=code.size();
        emit("JPC",0,0);
        output_index--;
        if (SYM_WORD=="do")
        {
            output[output_index]+="do ";
            output[output_index]+="Statement\t";
            output_index++;
            tree+="\"Terminal_1\":\"";
            tree+=SYM_WORD;
            tree+="\",";
            ADVANCE();
            tree+="\"Statement\":{";
            STATEMENT();
            tree+="}";
            emit("JMP",0,index0);
            code[index].a=code.size();
            output_index--;
        }
        else ERROR();
    }
    else
        ERROR();
}

void READ()
```

```
{
    if (Error)return;
    is_read=true;
    if (SYM_WORD=="read")
    {
        tree+="\"Terminal_0\":\"";
        tree+=SYM_WORD;
        tree+="\",";
        output[output_index]+="read ";
        ADVANCE();
        if (SYM_WORD=="(")
        {
            tree+="\"Terminal_1\":\"";
            tree+=SYM_WORD;
            tree+="\",";
            output[output_index]+="(";
            output[output_index]+="Identifier IdentifierA";
            output_index++;
            ADVANCE();
            tree+="\"ID\":";
            IDENTIFIER();
            tree+=",\"ID Any\":{";
            IDENTIFIER_ANY();
            tree+="},";
            output_index--;
            if (SYM_WORD==")")
            {
                tree+="\"Terminal_2\":\"";
                tree+=SYM_WORD;
                tree+="\"";
                output[output_index]+=")\t";
                ADVANCE();
            }
            else  ERROR();
        }
    }
    else ERROR();
    is_read=false;
}

void WRITE()
{
    is_write=true;
    if (Error)return;
```

```
    if (SYM_WORD=="write")
    {
        tree+="\"Terminal_0\":\"";
        tree+=SYM_WORD;
        tree+="\",";
        output[output_index]+="write";
        ADVANCE();
        if (SYM_WORD=="(")
        {
            tree+="\"Terminal_1\":\"";
            tree+=SYM_WORD;
            tree+="\",";
            output[output_index]+="( Expression ExpressionA ";
            output_index++;
            ADVANCE();
            tree+="\"Expression\":{";
            EXPRESSION();
            tree+="},\"Expression Any\":{";
            EXPRESSION_ANY( );
            tree+="},";
            output_index--;
            if (SYM_WORD==")")
            {
                tree+="\"Terminal_2\":\"";
                tree+=SYM_WORD;
                tree+="\"";
                output[output_index]+=")\t";
                ADVANCE();
            }
            else  ERROR();
        }
    }
    else ERROR();
    is_write=false;
}

bool is_relational(string s)
{
    return (s=="="||s=="#"||s=="<"||s=="<="||s==">"||s==">=");
}
```

3　解释执行

```
#include <iostream>
```

```
#include <fstream>
#include <stack>
#include <vector>

using namespace std;

typedef struct CODE
{
    string f;
    int l;
    int a;
} Code;
vector<Code> code;

fstream CODE("code.txt");
string
opra[]={"ret","+","-","*","/","#","<","<=",">",">=","=","%","read","w
rite"};
vector<int> Stack;
int P=0;
Code I;
int T;
int B;

void exec(Code I);
void output();

int main()
{
    string opr;
    int l,a;
    while (!CODE.eof())
    {
        Code tem;
        CODE>>tem.f;
        CODE>>tem.l;
        CODE>>tem.a;
        code.push_back(tem);
    } //read code
    /************** jump to the entry and initialize **************/
    P=code[0].a; I=code[P];
    P++; B=0;
    for (int i=0;i<I.a;i++)Stack.push_back(0);
    T=I.a;
```

```
    Stack[0]=0;
    Stack[1]=0;
    Stack[2]=0;

/*****************************************************************/
    while (!(B==0&&T==0))
    {
        I=code[P];
        P++;
        output();
        exec(I);
    }
    output();
    return 0;
}

void output()
{
    cout<<"P:"<<P<<"\tT:"<<T<<"\tB:"<<B<<"\tI:"<<I.f<<" "<<I.l<<"
"<<I.a<<"\t";
    for (int i=0;i<T;i++)
        cout<<Stack[i]<<" ";
    cout<<endl;
}

void exec(Code I)
{
    if (I.f=="LIT")
    {
        if (Stack.size()>T)
            Stack[T]=I.a;
        else Stack.push_back(I.a);
        T++;
    }else
    if (I.f=="LOD")
    {
        int base=B;
        for (int i=0;i<I.l;i++)
            base=Stack[base];
        if (Stack.size()>T)
            Stack[T]=Stack[base+I.a];
        else Stack.push_back(Stack[base+I.a]);
        T++;
    }else
```

```
if (I.f=="STO")
{
    int base=B;
    for (int i=0;i<I.l;i++)
        base=Stack[base];
    T--;
    Stack[base+I.a]=Stack[T];
}else
if (I.f=="CAL")
{
    int ret=P;
    int level=I.l;
    P=I.a;
    I=code[P];
    P++;
    for (int i=0;i<I.a;i++)
    {
        if (Stack.size()>T) Stack[T]=0;
        else Stack.push_back(0);
        T++;
    }
    if (level==0)Stack[T-I.a]=B;
    else if (level==1)Stack[T-I.a]=Stack[B];
    Stack[T-I.a+1]=B;
    Stack[T-I.a+2]=ret;
    B=T-I.a;
}else
if (I.f=="INT")
{
    /***NULL***/
}else
if (I.f=="JMP")
{
    P=I.a;
}else
if (I.f=="JPC")
{
    T--;
    if (!Stack[T]) P=I.a;
}else
if (I.f=="OPR")
{
    int base;
    switch (I.a)
```

```
{
case 0:
    P=Stack[B+2];
    base=B;
    B=Stack[B+1];
    T=base;
    break;
case 1:
    T-=1;
    Stack[T-1]=Stack[T-1]+Stack[T];
    break;
case 2:
    T-=1;
    Stack[T-1]=Stack[T-1]-Stack[T];
    break;
case 3:
    T-=1;
    Stack[T-1]=Stack[T-1]*Stack[T];
    break;
case 4:
    T-=1;
    Stack[T-1]=Stack[T-1]/Stack[T];
    break;
case 5:
    T-=1;
    if (Stack[T-1]!=Stack[T]) Stack[T-1]=1;
    else Stack[T-1]=0;
    break;
case 6:
    T-=1;
    if (Stack[T-1]<Stack[T]) Stack[T-1]=1;
    else Stack[T-1]=0;
    break;
case 7:
    T-=1;
    if (Stack[T-1]<=Stack[T]) Stack[T-1]=1;
    else Stack[T-1]=0;
    break;
case 8:
    T-=1;
    if (Stack[T-1]>Stack[T]) Stack[T-1]=1;
    else Stack[T-1]=0;
    break;
case 9:
```

```
            T-=1;
            if (Stack[T-1]>=Stack[T]) Stack[T-1]=1;
            else Stack[T-1]=0;
            break;
        case 10:
            T-=1;
            if (Stack[T-1]==Stack[T]) Stack[T-1]=1;
            else Stack[T-1]=0;
            break;
        case 11:
            T-=1;
            Stack[T-1]=Stack[T-1]%Stack[T];
            break;
        case 12:
            if (Stack.size()<=T) Stack.push_back(0);
            cin>>Stack[T];
            T++;
            break;
        case 13:
            T--;
            cout<<Stack[T]<<endl;
            break;
        }
    }
}
```

4 测试用例

```
const c1=2;
var v1,v2,v3,v4;
procedure p1;
 var v5;
 begin
  v5:=2;
  write(v5/2+2-1);
  while v3#0 do
  begin
   v4:=v1/v2;
   v3:=v1-v4*v2;
   v1:=v2;
   v2:=v3;
  end;
 end;
procedure p2;
```

```
 const c2=2;
 procedure p3;
  begin
   if v1#1 then
   begin
    v1:=v1-1;
    v2:=v2*v1;
    call p3;
   end;
  end;
 begin
  call p3;
  if odd c2 then
   write(c2);
  if c2=2 then
   write(c2+1)
 end;
begin
 read(v1,v2);
 if v1<v2 then
 begin
  v3:=v1;
  v1:=v2;
  v2:=v3;
 end;
 begin;
  v3:=1;
  call p1;
  write(c1,c1*v1,c1*v1/2);
 end;
 read(v1);
 v2:=v1;
 call p2;
 write(v2);
end.
```