计算机学院编译原理与技术课程实验报告

实验题目：语法分析		学号：201720130148
日期：	班级： 17.2	姓名：陈加乐
Email：1345068110@qq.com		

实验步骤与内容：

语法分析器的定义：

语法分析器（Parser）通常是作为编译器或解释器的组件出现的，
它的作用是进行语法检查、并构建由输入的单词组成的数据结构 （一般是语法分析树、抽象语法树等层次化的数据结构）。
语法检查，检查程序是否符合文法规则
语法分析器通常使用一个独立的词法分析器从输入字符流中分离出一个个的“单词”，并将单词流作为其输入。

首先实验所用的 PL0 文法的表示是 BNF（巴科斯范式）表示法
BNF（巴科斯范式）的表示规则：

在双引号中的字("word")代表着这些字符本身。而 double_quote 用来代表双引号。
在双引号外的字（有可能有下划线）代表着语法部分。
尖括号(＜＞)内包含的为必选项。
方括号([])内包含的为可选项。
大括号({})内包含的为可重复 0 至无数次的项。
竖线(|)表示在其左右两边任选一项，相当于"OR"的意思。
:= 是“被定义为”的意思。

	非终结符：		11	表达式	BDS
0	程序	P	12	项	TERM
1	分程序	SP	13	因子	FAC
2	常量说明部分	C	14	条件语句	COND
3	常量定义	CD	15	过程调用语句	PROCALL
4	变量说明部分	V	16	当型循环语句	LOOP
5	过程说明部分	PRO	17	读语句	DU
6	过程首部	PRPHEAD	18	写语句	XIE
7	语句	SEN	19	加减运算符	PS
8	赋值语句	ASS	20	乘除运算符	MULD
9	复合语句	FH	21	关系运算符	RELA
10	条件	CDI			

	终结符：	12	Read	
1	Const	13	Write	
2	Var	14	： =	
3	procedure	15	,	
4	Begin	16	;	
5	End	17	无符号整数	usNum
6	Odd	18	标识符	Id
7	if	19		
8	Then	20		
9	call	21		
10	While	22	字母	Alpha
11	Do	23	数字	Number

〈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〉

〈变量说明部分〉→VAR〈标识符〉{,〈标识符〉};

〈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〉)

〈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〉})

〈无符号整数〉→〈数字〉{〈数字〉}

〈标识符〉→〈字母〉{〈字母〉|〈数字〉}

〈加减运符〉→+|-

〈乘除运算符〉→*|/

〈关系运算符〉→=|#|<|<=|>|>=

〈字母〉→a|b|c…x|y|z

〈数字〉→0|1|2…7|8|9

0:P->SP

1:SP->[C] [V] [PRO] SEN

2:C->const CD{,CD};

3:CD->id = usnum

4:V->var id{,id};

5:PRO->PRPHEAD SP;{PRO}

6:PRPHEAD->procedure id;

7:SEN->ASS|FH|COND|LOOP|PROCALL|DU|XIE|空

8:ASS->id := BDS

9: FH->begin
SEN{;SEN}end

10: CDI->BDS RELA BDS|
odd BDS

11: BDS->[+|-]TERM
{PS TERM}

12:TERM->FAC {MULD FAC}

13:FAC->id|num|(BDS)

14:COND->if CDI then SEN

15:PROCALL->call id

16:LOOP->while CDI do SEN

17:DU->read(id {,id})

18:XIE->write(id{,id})

19：PS->+|-

20:MULD->*|/

21:RELA->=|#|<|<=|>|>=

一开始打算使用自下而上的 LR(1)分析法；
观察文法可得，短时间内使用 LR（1）分析法较为困难
所以这里使用递归下降法进行语法分析
对于每一个非终结符，根据其对应的产生式，给出其分程序
首先是分程序的声明：

```
//声明所有子程序
void P(); //主程序
void SP(); //分程序
void C(); //常量说明部分
void CD();//常量定义
void V(); //变量说明部分
void PRO(); //过程说明部分
void PRPHEAD();//过程头
void SEN(); //语句
void ASS();//赋值语句
void FH(); //复合语句
void CDI(); //条件
void BDS(); //表达式
void TERM();//项
void FAC(); //因子
void COND(); //条件调用语句
void PROCALL(); //过程调用语句
void LOOP();//当型循环语句
void DU(); //读语句
void XIE(); //写语句
void PS();//加减运算符
void MULD();//乘除运算符
void RELA();//关系运算符
```

语法分析错误提示函数：

```
//语法分析，错误提示函数
void parserror(){
    cout<<"pasing error!!!\n";
    cout<<"< ";
        cout.width(3);  cout<<cur->sn<<",   ";
        cout.width(10);  cout<<kw[cur->k]<<"    ,   ";
        cout.width(5);  cout<<cur->k<<" ,   ";
        cout.width(5);
    if(cur->k==constant &&strtoi(cur->v)!=-1)
        cout<<num[strtoi(cur->v)] <<" >\n";
    else cout<<cur->v <<" >\n";
    exit(0);
}
```

定义一个迭代器，
顺序访问词法分析得到的单词序列：

```
seq* cur;
```

输出结果：
举例说明，若给出一个有误的 PL0 源程序：

```
const a=10;
var d,e,f;
procedure p;
var g;
begin
  d:=a*2;
  e:=a//3;
  if d<=e then f:=d+e
end;
begin
  read(e,f);
  write(e,f,d);
  call p;
  while odd d do e:=-e+1
end.
```

输出结果：

```
<  25,  identifier  ,  29  ,  e >
<  26,         :=    ,  15  ,  >
<  27,  identifier  ,  29  ,  a >
<  28,          /    ,  19  ,  >
<  29,          /    ,  19  ,  >
<  30,   constant    ,  30  ,  3 >
<  31,          ;    ,  26  ,  >
<  61,        call   ,   9  ,  >
<  62,  identifier   ,  29  ,  p >
<  63,          ;    ,  26  ,  >
<  64,       while   ,  10  ,  >
<  65,         odd   ,   6  ,  >
<  66,  identifier   ,  29  ,  d >
<  67,          do   ,  11  ,  >
<  68,  identifier   ,  29  ,  e >
<  69,          :=   ,  15  ,  >
<  70,          -    ,  17  ,  >
<  71,  identifier   ,  29  ,  e >
<  72,          +    ,  16  ,  >
<  73,   constant    ,  30  ,  1 >
<  74,        end    ,   5  ,  >
<  75,               ,  -1  ,  >
pasing error!!!
<  29,          /    ,  19  ,  >
```

可以看出确实分析出了语法错误，并指出错误位置
结论分析与体会： 暂时只实现了，检测源程序是否出现语法错误， 关于语义错误的信息，这里并未进行检查，下一步将进行这一操作

源程序：

这里把上次的实验代码，拆分成，PL0.c 主程序和 PL0.h 头文件

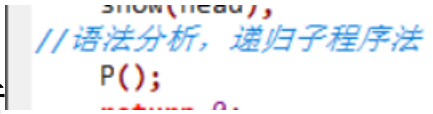
实验程序结构：

Main 程序,一步步对源程序进行编译

PL0.h，给出词法分析，以及基础的数据结构的定义

Parse.h 编译时进行语法分析的相关程序

Main 程序基本与上次实验相同，



增加了语法分析

```
#include<iostream>
 //#include "pl0.h"
#include "parse.h"

char *name="E:\\My_project\\Lesson\\compiler_PTT\\lab\\PL0_code\\PL0_code0.in";
int main()
{
/* */
//创建保留字表reserve(该表同样兼具单词编码表)，还有编码对应单词表kw
    create_keytable();
    //读取PL0源程序文件
    string t=fr(name);
    cout<<t<<endl;
    //    t=preprocessing(t);//对源程序做预处理
    //  show_word_coding_table();//屏幕输出单词类别表
//词法分析程序，产生单词序列链表，head
    lexer(t);
    //输出单词序列链表head
    show(head);
//语法分析，递归子程序法
    P();
    return 0;
}
```

PL0.h 头文件与上一次实验报告基本没变化

新写的 parse.h 如下：

Parse.h:

```
/*
递归下降法，语法分析
依据各产生式，定义递归子程序

输入，词法分析产生的单词序列
head
*/
#include "pl0.h"
//访问单词序列的迭代器
seq* cur;

//创建语法树的存储结构
struct parsetree{
    string name;//非终结符
    seq* node;//单词符号指针   （叶子节点才是单词）
    int level;//节点所在层次
    parsetree* father;
    parsetree** child;
}*root;
//语法树的遍历，后序遍历（左右根）
void postorder(){

}

//声明所有子程序
void P(); //主程序
void SP(); //分程序
void C(); //常量说明部分
void CD();//常量定义
void V(); //变量说明部分
void PRO(); //过程说明部分
void PRPHEAD();//过程头
void SEN(); //语句
void ASS();//赋值语句
void FH(); //复合语句
void CDI(); //条件
void BDS(); //表达式
void TERM();//项
void FAC(); //因子
void COND(); //条件调用语句
void PROCALL(); //过程调用语句
void LOOP();//当型循环语句
void DU(); //读语句
void XIE(); //写语句
```

```
void PS();//加减运算符
void MULD();//乘除运算符
void RELA();//关系运算符

//语法分析，错误提示函数
void parserror(){
    cout<<"pasing error!!!\n";
    cout<<"< ";
        cout.width(3);  cout<<cur->sn<<",  ";
        cout.width(10);  cout<<kw[cur->k]<<"  ,   ";
        cout.width(5);  cout<<cur->k<<",   ";
        cout.width(5);
    if(cur->k==constant &&strtoi(cur->v)!=-1)
        cout<<num[strtoi(cur->v)] <<" >\n";
    else cout<<cur->v <<" >\n";
    exit(0);
}

void P(){//主程序
    cur=head;
    SP();
    cout<<"ok!!!\n";
}
void SP(){//分程序
    if(kw[cur->k] =="const")//存在常量说明部分
        C();
    if(kw[cur->k]=="var")//存在变量说明部分
        V();
    if(kw[cur->k]=="procedure")//存在过程说明部分
        PRO();
    SEN();
    //规约
}
void C(){//常量说明部分
    if(kw[cur->k] =="const"){
        cur=cur->next;
        CD();
        while(kw[cur->k]==",")
            CD();
        if(kw[cur->k]==";")
            cur=cur->next;
        else parserror();
    }else parserror();
}
```

```
void CD(){//常量定义
    if(kw[cur->k]=="identifier"){
        cur=cur->next;
        if(kw[cur->k]=="=")  {
            cur=cur->next;
            if(kw[cur->k]=="constant"){
                cur=cur->next;
            }else parserror();
        }else parserror();
    }else parserror();
}

void V(){//变量说明部分 ,关于变量层次说明，放在语义分析时再对程序进行修改
    if(kw[cur->k]=="var"){
        cur=cur->next;
        if(kw[cur->k]=="identifier"){
            cur=cur->next;
            while(kw[cur->k]==","){
                    cur=cur->next;
                if(kw[cur->k]=="identifier")
                    cur=cur->next;
                else parserror();
            }
            if(kw[cur->k]==";")
                cur=cur->next;
            else parserror();
        }else parserror();
    }else parserror();
}

void PRO(){//过程说明部分
    PRPHEAD();
    SP();
    if(kw[cur->k]==";")
        cur=cur->next;
    else parserror();
    while(kw[cur->k]=="procedure")
            PRO();
}

void PRPHEAD(){//过程首部
    if(kw[cur->k]=="procedure"){
        cur=cur->next;
        if(kw[cur->k]=="identifier"){
```

```
            cur=cur->next;
            if(kw[cur->k]==";"){
                cur=cur->next;
            }else parserror();
        }else parserror();
    }else parserror();

}

void SEN(){//语句
    if(kw[cur->k]=="identifier"){
        ASS();
    }else if(kw[cur->k]=="begin"){
        FH();
    }else if(kw[cur->k]=="if"){
        COND();
    }else if(kw[cur->k]=="call"){
        PROCALL();
    }else if(kw[cur->k]=="while"){
        LOOP();
    }else if(kw[cur->k]=="read"){
        DU();
    }else if(kw[cur->k]=="write"){
        XIE();
    }else cout<<"kong..."<<kw[cur->k]<<"...."<<endl;
}

void ASS(){//赋值语句
    if(kw[cur->k]=="identifier"){
        cur=cur->next;
        if(kw[cur->k]==":="){
            cur=cur->next;
            BDS();
        }else parserror();
    }else parserror();
}

void FH(){//复合语句
    if(kw[cur->k]=="begin"){
        cur=cur->next;
        SEN();
        while(kw[cur->k]==";"){
            cur=cur->next;
            SEN();
```

```
        }
        if(kw[cur->k]=="end"){
            cur=cur->next;
        }else parserror();
    }else parserror();
}

void CDI(){//条件
    if(kw[cur->k]=="odd"){
        cur=cur->next;
        BDS();
    }else {
        BDS();
        RELA();
        BDS();
    }
}

void BDS(){//表达式
    if(kw[cur->k]=="+"){
        cur=cur->next;
    }else if( kw[cur->k]=="-"){
        cur=cur->next;
    }
    TERM();
    while(kw[cur->k]=="+"| kw[cur->k]=="-"){
        PS();
        TERM();
    }
}

void TERM(){//项
    FAC();
    while(kw[cur->k]=="*"| kw[cur->k]=="/"){
        MULD();
        FAC();
    }
}

void FAC(){//因子
    if(kw[cur->k]=="identifier"){
        cur=cur->next;
    }else if(kw[cur->k]=="constant"){
        cur=cur->next;
```

```
    }else if(kw[cur->k]=="("){
        BDS();
        if(kw[cur->k]==")")
            cur=cur->next;
        else parserror();
    }else parserror();
}

void COND(){//条件语句
    if(kw[cur->k]=="if"){
        cur=cur->next;
        CDI();
        if(kw[cur->k]=="then"){
            cur=cur->next;
            SEN();
        }else parserror();
    }else parserror();
}

void PROCALL(){//过程调用语句
    if(kw[cur->k]=="call"){
        cur=cur->next;
        if(kw[cur->k]=="identifier")
            cur=cur->next;
        else parserror();
    }else parserror();
}

void LOOP(){//当型循环语句
    if(kw[cur->k]=="while"){
        cur=cur->next;
        CDI();
        if(kw[cur->k]=="do"){
            cur=cur->next;
            SEN();
        }else parserror();
    }else parserror();
}

void DU(){//读语句
    if(kw[cur->k]=="read"){
        cur=cur->next;
        if(kw[cur->k]=="("){
            cur=cur->next;
```

```
            if(kw[cur->k]=="identifier"){
                cur=cur->next;
                while(kw[cur->k]==","){
                    cur=cur->next;
                    if(kw[cur->k]=="identifier"){
                        cur=cur->next;
                    }else parserror();
                }
                if(kw[cur->k]==")"){
                    cur=cur->next;
                }else parserror();
            }else parserror();
        }else parserror();
    }else parserror();
}
void XIE(){//写语句
    if(kw[cur->k]=="write"){
        cur=cur->next;
        if(kw[cur->k]=="("){
            cur=cur->next;
            if(kw[cur->k]=="identifier"){
                cur=cur->next;
                while(kw[cur->k]==","){
                    cur=cur->next;
                    if(kw[cur->k]=="identifier"){
                        cur=cur->next;
                    }else parserror();
                }
                if(kw[cur->k]==")"){
                    cur=cur->next;
                }else parserror();
            }else parserror();
        }else parserror();
    }else parserror();
}

void PS(){//加减运算符
    if(kw[cur->k]=="+"){
        cur=cur->next;
    }else if(kw[cur->k]=="-"){
        cur=cur->next;
    }else parserror();
}
void MULD(){//乘除运算符
```

```
    if(kw[cur->k]=="*"){
        cur=cur->next;
    }else if(kw[cur->k]=="/"){
        cur=cur->next;
    }else parserror();
}
void RELA(){//关系运算符
    if(kw[cur->k]=="="){
        cur=cur->next;
    }else if(kw[cur->k]=="#"){
        cout<<"q"<<cur->sn<<endl;
        cur=cur->next;
    }else if(kw[cur->k]==">"){
        cur=cur->next;
    }else if(kw[cur->k]==">="){
        cur=cur->next;
    }else if(kw[cur->k]=="<"){
        cur=cur->next;
    }else if(kw[cur->k]=="<="){
        cur=cur->next;
    }else parserror();
}
```