실험 3 : 구문 분석기

설명

이 과제는 주어진 문법 문서와 가이드북을 참고하여 구문 분석 프로그램을 설계하고 구현해야 합니다. 소스 코드에서 해당하는 구문 요소를 식별하는데 있어 다음과 같은 입력, 출력 및 처리 요구사항이 있습니다:

1. 제시된 문법 규칙에 따라 재귀 하위 프로그램 방법을 사용하여 정의된 구문 요소를 분석해야 합니다.
2. 어휘 분석을 통해 식별된 단어의 순서에 따라, 각 단어의 정보를 한 줄에 하나씩 출력해야 합니다. (예: 단어 유형 코드 단어 문자열 (간격은 하나의 공백으로 구분))
3. 문법에서 등장하는 (<BlockItem>, <Decl>, <BType>을 제외한) 구문 분석 요소가 분석되기 전에, 현재 구문 요소의 이름을 새로운 줄로 출력해야 합니다. (예: "<Stmt>")
4. 참고: 출력이 요구되지 않는 구문 요소들도 분석을 수행해야 하지만 출력은 필요하지 않습니다.

문법

```
 // 0-> 1.是否存在Decl 2.是否存在FuncDef
编译单元 CompUnit → {Decl} {FuncDef} MainFuncDef

// 1<-0 覆盖两种声明
声明 Decl → ConstDecl | VarDecl
基本类型 BType → 'int' // 存在即可

//ConstDecl부분
// 2<-1 1.花括号内重复0 次 2.花括号内重复多次
常量声明 ConstDecl → 'const' BType ConstDef { ',' ConstDef } ';'
// 3<-2 변수, 1차원, 2차원 배열을 포함
常数定义 ConstDef → Ident { '[' ConstExp ']' } '=' ConstInitVal
// 4<-3 위에서 선언한 변수, 1차원, 2차원 배열을 초기화
常量初值 ConstInitVal → ConstExp | '{' [ ConstInitVal { ',' ConstInitVal } ] '}'

// 5<-1 1花括号内重复0次 2.花括号内重复多次
变量声明 VarDecl → BType VarDef { ',' VarDef } ';'
// 包含普通变量、一维数组、二维数组定义
变量定义 VarDef → Ident { '[' ConstExp ']' } | Ident { '[' ConstExp ']' } '=' InitVal
// 1.表达式初值 2.一维数 组初值 3.二维数组初值
变量初值 InitVal → Exp | '{' [ InitVal { ',' InitVal } ] '}'


函数定义 FuncDef → FuncType Ident '(' [FuncFParams] ')' Block // 1.无形参 2.有形参

// 0. 存在main函数
主函数定义 MainFuncDef → 'int' 'main' '(' ')' Block
// 覆盖两种类型的函数
函数类型 FuncType → 'void' | 'int'
// 1. 1.花括号内重复0次 2.花括号内重复多次
函数形参表 FuncFParams → FuncFParam { ',' FuncFParam }
// 1.普通变量 2.一维数组变量 3.二维数组变量<- 이거안함
函数形参 FuncFParam → BType Ident ['[' ']' { '[' ConstExp ']' }]



// 1.花括号内重复0次 2.花括号内重复多次
语句块 Block → '{' { BlockItem } '}'
// 覆盖两种语句块项
语句块项 BlockItem → Decl | Stmt
```

```
// 每种类型的语句都要覆盖
语句 Stmt →
LVal '=' Exp ';'
| [Exp] ';' //有无Exp两种情况
| Block
| 'if' '(' Cond ')' Stmt [ 'else' Stmt ] // 1.有else 2.无else
//1. 无缺省 2. 缺省第一个 ForStmt 3. 缺省Cond 4. 缺省第二个ForStmt
| 'for' '(' [ForStmt] ';' [Cond] ';' [ForStmt] ')' Stmt
| 'break' ';' | 'continue' ';'
| 'return' [Exp] ';' // 1.有Exp 2.无Exp
| LVal '=' 'getint''('')'';'
| 'printf''('FormatString{','Exp}')'';' // 1.有Exp 2.无Exp

语句 ForStmt → LVal '=' Exp // 存在即可
表达式 Exp → AddExp 注:SysY 表达式是int 型表达式 // 存在即可
条件表达式 Cond → LOrExp // 存在即可
// 1.普通变量 2.一维数组 3.二维数组
左值表达式 LVal → Ident {'[' Exp ']'}

// 3
// 三种情况均需覆盖
基本表达式 PrimaryExp → '(' Exp ')' | LVal | Number
数值 Number → IntConst // 存在即可

// 3种情况均需覆盖,函数调用也需要覆盖FuncRParams的不同情况
一元表达式 UnaryExp → PrimaryExp | Ident '(' [FuncRParams] ')' | UnaryOp UnaryExp // 存在即可
单目运算符 UnaryOp → '+' | '−' | '!' 注:'!'仅出现在条件表达式中 // 三种均需覆盖
// 1.花括号内重复0次 2.花括号内重复多次 3.Exp需要覆盖数组传参和部分数组传参
函数实参表 FuncRParams → Exp { ',' Exp }


// 1// 1.UnaryExp 2.* 3./ 4.% 均需覆盖
乘除模表达式 MulExp → UnaryExp | MulExp ('*' | '/' | '%') UnaryExp
加减表达式 AddExp → MulExp | AddExp ('+' | '−') MulExp // 1.MulExp 2.+ 需覆盖 3.- 需 覆盖

关系表达式 RelExp → AddExp | RelExp ('<' | '>' | '<=' | '>=') AddExp // 1.AddExp 2.< 3.> 4.<= 5.>= 均需覆盖
相等性表达式 EqExp → RelExp | EqExp ('==' | '!=') RelExp // 1.RelExp 2.== 3.!= 均需 覆盖

逻辑与表达式 LAndExp → EqExp | LAndExp '&&' EqExp // 1.EqExp 2.&& 均需覆盖
逻辑或表达式 LOrExp → LAndExp | LOrExp '||' LAndExp // 1.LAndExp 2.|| 均需覆盖
常量表达式 ConstExp → AddExp 注:使用的Ident 必须是常量 // 存在即可
```

입력 형식

문법 이해 파일에 있는 **testfile.txt에는 문법 요구 사항을 준수하는 테스트 프로그램이 있습니다.**

출력 형식

어휘 분석 결과를 output.txt에 출력해야 합니다.

입력 예시

```
int main(){
    int c;
    c= getint();
    printf("%d",c);
    return c;
}
```

출력 예시

```
INTTK int
MAINTK main
LPARENT (
RPARENT )
LBRACE {
INTTK int
IDENFR c
<VarDef>
SEMICN ;
<VarDecl>
IDENFR c
<LVal>
ASSIGN =
GETINTTK getint
LPARENT (
RPARENT )
SEMICN ;
<Stmt>
PRINTFTK printf
LPARENT (
STRCON "%d"
COMMA ,
IDENFR c
<LVal>
<PrimaryExp>
<UnaryExp>
<MulExp>
<AddExp>
<Exp>
RPARENT )
SEMICN ;
<Stmt>
RETURNTK return
IDENFR c
<LVal>
<PrimaryExp>
<UnaryExp>
<MulExp>
<AddExp>
<Exp>
SEMICN ;
<Stmt>
RBRACE }
<Block>
<MainFuncDef>
<CompUnit>
```