실험 4 : 의미 분석 + 오류 출력

문제 설명

결과 파일에는 다음과 같은 두 가지 정보가 포함되어 있습니다: 오류가 있는 행 번호, 잘못된 카테고리 코드 (행 번호와 카테고리 코드 사이에 공백만 있고 카테고리 코드는 표의 소문자 영문자를 엄격히 따름)

오류 범주 코드는 다음 표에 정의된 대로 출력되며 행 번호는 1부터 계산됩니다.

错误类型	错误类别码	解释	对应文法及出错符号 (... 表示省略该条规则后续部分)
非法符号	a	格式字符串中出现非法字符报错行号为 **<FormatString>** 所在行数。	<FormatString> → '"'{<Char>}'"'
名字重定义	b	函数名或者变量名在**当前作用域**下重复定义。注意，变量一定是同一级作用域下才会判定出错，不同级作用域下，内层会覆盖外层定义。报错行号为 **<Ident>** 所在行数。	<ConstDef>→<Ident> ... <VarDef>→<Ident> ... <Ident> ... <FuncDef>→<FuncType><Ident> ... <FuncFParam> → <BType> <Ident> ...
未定义的名字	c	使用了未定义的标识符报错行号为 **<Ident>** 所在行数。	<LVal>→<Ident> ... <UnaryExp>→<Ident> ...
函数参数个数不匹配	d	函数调用语句中，参数个数与函数定义中的参数个数不匹配。报错行号为函数调用语句的**函数名**所在行数。	<UnaryExp>→<Ident>'('[<FuncRParams>]')'
函数参数类型不匹配	e	函数调用语句中，参数类型与函数定义中对应位置的参数类型不匹配。报错行号为函数调用语句的**函数名**所在行数。	<UnaryExp>→<Ident>'('[<FuncRParams>]')'
无返回值的函数存在不匹配的return语句	f	报错行号为 **'return'** 所在行号。	<Stmt>→'return' {'['<Exp>']'}';'
有返回值的函数缺少return语句	g	只需要考虑函数末尾是否存在return语句，**无需考虑数据流**。报错行号为函数**结尾的'}'** 所在行号。	<FuncDef> → <FuncType> <Ident> '(' [<FuncFParams>] ')' <Block> <MainFuncDef> → 'int' 'main' '(' ')' <Block>
不能改变常量的值	h	<LVal>为常量时，不能对其修改。报错行号为 **<LVal>** 所在行号。	<Stmt>→<LVal>'=' <Exp>';' <Stmt>→<LVal>'=' 'getint' '(' ')' ';'
缺少分号	i	报错行号为分号**前一个非终结符**所在行号。	<Stmt>,<ConstDecl>及<VarDecl>中的';'
缺少右小括号')'	j	报错行号为右小括号**前一个非终结符**所在行号。	函数调用(<UnaryExp>)、函数定义(<FuncDef>)及<Stmt>中的')'
缺少右中括号']'	k	报错行号为右中括号**前一个非终结符**所在行号。	数组定义(<ConstDef>,<VarDef>,<FuncFParam>)和使用(<LVal>)中的']'
printf中格式字符与表达式个数不匹配	l	报错行号为 **'printf'** 所在行号。	<Stmt> →'printf''('<FormatString>{,<Exp>}')'';'
在非循环块中使用break和continue语句	m	报错行号为 **'break'** 与 **'continue'** 所在行号。	<Stmt>→'break'';' <Stmt>→'continue'';'

오류 i, j, k타입의 "前一个非终结符"는 문법 규칙의 ;)] 이전의 비종결 기호를 강조합니다.

분석에서는 비종결 기호에 의해 생성된 마지막 기호, 즉 ;] 정상적으로 나타나야 할 위치의 이전 단어를 처리합니다.

(2) 문자, 문자열의 줄바꿈 문자, 함수 호출 등을 포함한 모든 오류는 악의적인 줄바꿈을 하지 않습니다.

문법(오류 포함) 설명

```
 // 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 } ';'  // i
```

```
// 3<-2 변수, 1차원, 2차원 배열을 포함
常数定义 ConstDef → Ident { '[' ConstExp ']' } '=' ConstInitVal // b k
// 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 } ';' // i
// 包含普通变量、一维数组、二维数组定义
变量定义 VarDef → Ident { '[' ConstExp ']' } // b
| Ident { '[' ConstExp ']' } '=' InitVal //k
// 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 } ] '}'
 /*
 -> a. Exp : 1
 -> b. { Exp } : {1}
 -> 0 { Exp } : {}
 -> c. { Exp, Exp ... } : { 1, 2, 3 }
 -> d. { initVal } -> {{ InitVal }} -> {{Exp}} : {{1,2}}
 -> e.                {{ InitVal }} -> {{Exp,Exp...}} : {{1,2},{3,4}}
*/

// 0. 메인 함수
主函数定义 MainFuncDef → 'int' 'main' '(' ')' Block //g j

// 일반 함수
函数定义 FuncDef → FuncType Ident '(' [FuncFParams] ')' Block // b g j
// 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 ']' }] // b k



// 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 ';' h i
| [Exp] ';' //有无Exp两种情况 i
| Block //  i
| 'if' '(' Cond ')' Stmt [ 'else' Stmt ] // j
//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' ';' // i m
| 'return' [Exp] ';' // f i
| LVal '=' 'getint''('')'';' // h i j
| 'printf''('FormatString{','Exp}')'';' // i j l

语句 ForStmt → LVal '=' Exp // h
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 ']'}  // c k

// 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 func(func(1)); // 이런건 없다
一元表达式 UnaryExp → PrimaryExp | Ident '(' [FuncRParams] ')' // c d e j
```

```
 | UnaryOp UnaryExp //
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
表达式 Exp → AddExp 注:SysY 表达式是int 型表达式 // 存在即可

格式字符串:
<FormatString> → '"'{<Char>}'"' // a
```

입출력 예시

样例输入

```
const int const1 = 1, const2 = -100;
int change1;
int gets1(int var1,int var2){
    const1 = 999;
    change1 = var1 + var2          return (change1);
}
int main(){
    change1 = 10;
    printf("Hello World$");
    return 0;
}
```

样例输出

```
4 h
5 i
9 a
```

```
int main() {
    break;
    continue;
    return 0;
}
2 m
3 m


//2
void f1(int x, int y, int z){

}
```

```
int main() {
    f1(1);
    return 0;
}
6 d


//3
int main() {
    a;
    a = 1;
    a();
    return 0;
}
2 c
3 c
4 c


//4
void f(int a, int a){

}

int main() {
    int a, a;
    return 0;
}
1 b
6 b


//5
int main() {
    printf("#:\n");
    printf("%%");
    printf("\");
    return 0;
}
2 a
3 a
4 a


//6
int main() {
    printf("%d", 1, 1);
    return 0;
}
2 l


//7
int main() {
    int a[2][2;
    return 0;
}
2 k


//8
void f1({

}
```

```
void f2(){

}

int main() {
    f2(;
    return 0;
}
1 j
10 j


//9
int main() {
    return 0
}
2 i


//10
int main() {
    const int a = 0;
    a = 1;
    return 0;
}
3 h


//11
int f1()
{}

int main() {

}
2 g
6 g


//12
void f1(){
    return 0;
}

int main() {
    return 0;
}
2 f


//13
int f1(int x){
    return x;
}

void f2(int x[])
{
    return;
}

int main() {
    int arr[2][2];
    f2(1);
    f1(f2(arr[0]));
```

```
    return 0;
}
```