Compiler COMP032001

Parser / 2016112905 김민섭

차례

1. C- 문법 중 EBNF로 바꾼 부분

EBNF로 고쳤을 때 중괄호('{', '}')와 대괄호('[', ']')가 원래 문법과 중복되어 BNF 표현으로 바꿨습니다.

- Non-terminal들은 모두 화살 괄호('<', '>')로 둘러쌌습니다.
- Terminal들은 영대문자로 표시했습니다.
- 화살표는 '::='로 표시했습니다.

<declaration-list>	::=	<declaration> { <declaration> }
<param-list>	::=	<param> { COMMA <param> }
<param>	::=	<type-specifier> ID [LSBRACK RSBRACK]
<local-declarations>	::=	<var-declaration> { <var-declaration> }
	|	EMPTY
<statement-list>	::=	<statement> { <statement> }
	|	EMPTY
<expression-stmt>	::=	[<expression>] SEMI
<selection-stmt>	::=	IF LPAREN <expression> RPAREN <statement> [ELSE <statement>]
<return-stmt>	::=	RETURN [<expression>] SEMI
<var>	::=	ID [LSBRACK <expression> RSBRACK]
<simple-expression>	::=	<additive-expression> [<relop> <additive-expression>]
<additive-expression>	::=	<term> { <addop> <term>}
<term>	::=	<factor> { <mulop> <factor>}
<arg-list>	::=	<expression> { COMMA <expression> }

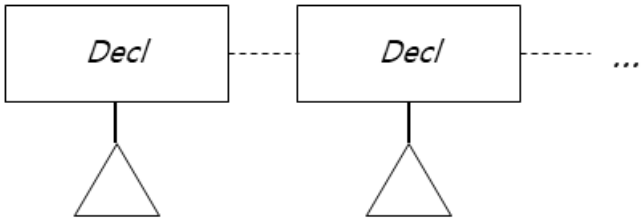
2. Syntax Structure for C-

2.1. Decl, VarDecl, FunDecl

문법

<program>	::=	<declaration-list>
<declaration-list>	::=	<declaration> { <declaration> }

다이어그램



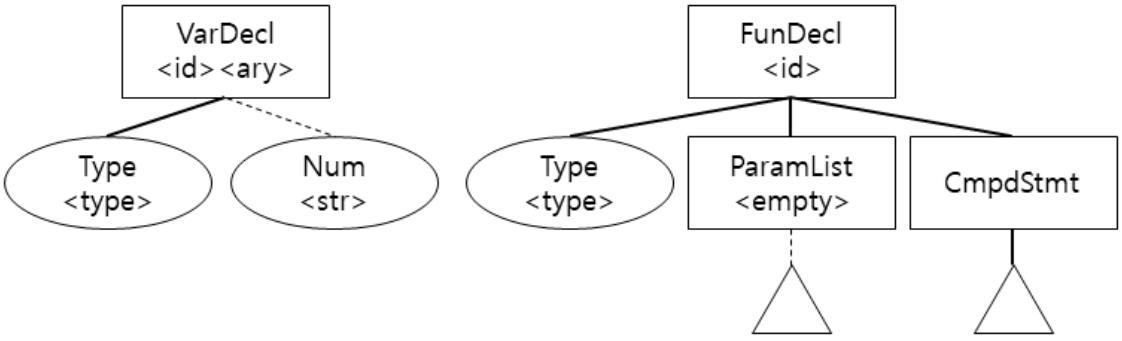
- 파서는 Program에 해당하는 노드 없이 곧바로 Sibling으로 연결된 첫번째 Decl 노드를 리턴합니다.
- Decl 노드는 노드 생성 시 임시로 할당될 수 있지만 곧 VarDecl이나 FunDecl로 바뀌게 됩니다. 이렇게 실제로 존재하지 않는 노드를 이탤릭체로 표시하였습니다.

2.1.2. VarDecl, FunDecl

문법

<declaration>	::=	<var-declaration>
	|	<fun-declaration>
<var-declaration>	::=	<type-specifier> ID SEMI
	|	<type-specifier> ID LSBRACK NUM RSBRACK SEMI
<fun-declaration>	::=	<type-specifier> ID LPAREN <params> RPAREN <compound-stmt>

다이어그램



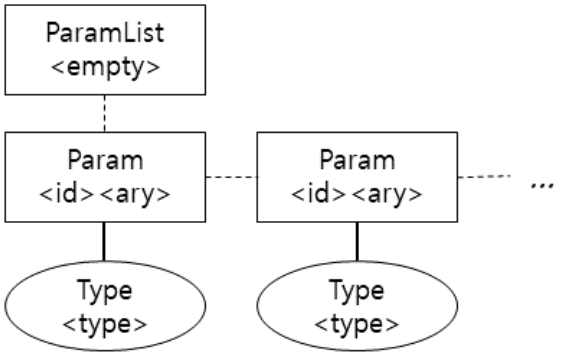
- VarDecl의 attribute는 string id와 bool ary입니다. 만약 배열 표현이 있다면 ary는 true값이 되고 두번째 자식으로 Num을 가집니다.
- FunDecl의 attribute는 string id입니다. 반드시 자식 노드로 Type, ParamList, CmpdStmt를 가집니다.
- Type을 노드로 구현한 이유는 언어가 확장될 때 타입을 정의하는 키워드가 복잡해질 수 있기 때문입니다.
- Num을 노드로 구현한 이유도 마찬가지로 언어가 확장될 때 다양한 리터럴 표현과 데이터 타입을 가질 수 있기 때문입니다.

2.2. ParamList, Param

문법

<params>	::=	<param-list>
	|	VOID
<param-list>	::=	<param> { COMMA <param> }
<param>	::=	<type-specifier> ID [LSBRACK RSBRACK]

다이어그램



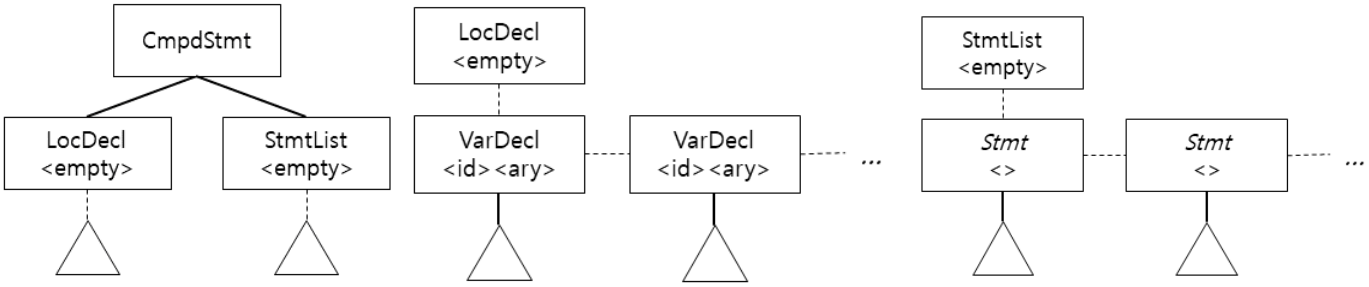
- ParamList의 bool empty는 함수 인자가 void일 경우 true가 되고, 그렇지 않다면 Param 노드를 자식 노드로 가집니다.
- Param의 attribute는 string id, bool ary입니다. 배열 표현이 있을 경우 ary 값은 true가 됩니다. 자식노드로 Type을 가지고 또다른 Param 노드를 sibling으로 가질 수 있습니다. sibling을 가졌는지 가지지 않았는지를 구분하는 방법은 노드 구조체의 sibling 포인터의 값이 nullptr인지 아닌지 확인합니다.

2.3. CmpdStmt, LocDecl, StmtList

문법

<compound-stmt>	::=	LCBRACK <local-declarations> <statement-list> RCBRACK
<local-declarations>	::=	<var-declaration> { <var-declaration> }
	|	EMPTY
<statement-list>	::=	<statement> { <statement> }
	|	EMPTY

다이어그램



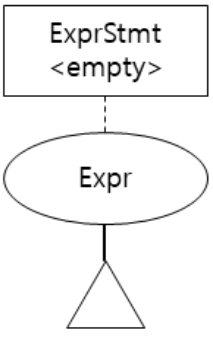
- CmpdStmt는 반드시 LocDecl, StmtList를 자식으로 갖습니다. FunDecl 노드의 관점에서 CmpdStmt 노드는 존재하지 않아도 되지만, StmtList의 자식으로 재귀적으로 나타날 수 있기 때문에 syntax 분석을 쉽게 하기 위해 구현해 놓았습니다.
- LocDecl은 자식이 없을 경우 bool empty의 값이 true가 됩니다. 자식으로 VarDecl을 가질 수 있으며 sibling으로 연결될 수 있습니다.
- StmtList도 마찬가지로 자식이 없을 경우 bool empty의 값은 true가 됩니다. 자식으로 Stmt를 가질 수 있으며, sibling으로 연결될 수 있습니다. Stmt 노드는 실제로는 존재하지 않지만 Statement 종류의 Nonterminal들을 나타내기 위해 사용했습니다.

2.4. ExprStmt

문법

```
<expression-stmt>      ::=  [ <expression> ] SEMI
```

다이어그램



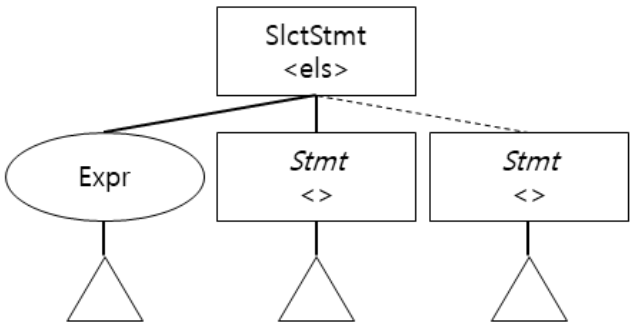
- ExprStmt는 자식이 없을 경우 bool empty의 값은 true가 됩니다.
- Args에서의 Expr는 sibling을 가질 수 있지만, 현재 문법에서는 sibling이 없습니다.

2.5. SlctStmt

문법

<selection-stmt>	::=	IF LPAREN <expression> RPAREN <statement> [ELSE <statement>]

다이어그램



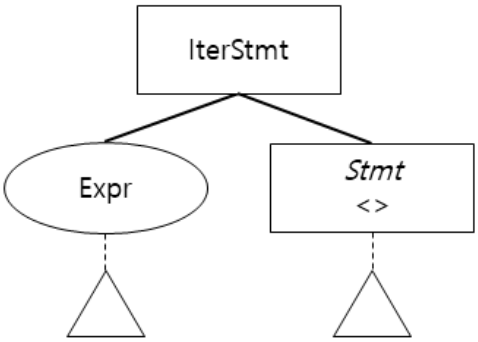
- SlctStmt는 if문에 해당하는 Expr, then문에 해당하는 Stmt를 자식으로 가집니다. else문이 있을 경우 bool els의 값은 true가 되고 세번째 자식으로 Stmt를 가집니다.

2.6. IterStmt

문법

<iteration-stmt>	::=	WHILE LPAREN <expression> RPAREN <statement>

다이어그램



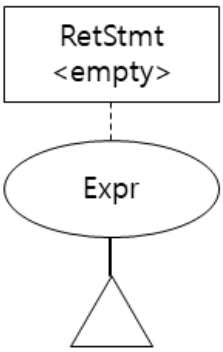
- IterStmt는 while에 해당하는 Expr, 반복문에 해당하는 Stmt를 자식으로 가집니다.

2.7. RetStmt

문법

<return-stmt>	::=	RETURN [<expression>] SEMI

다이어그램



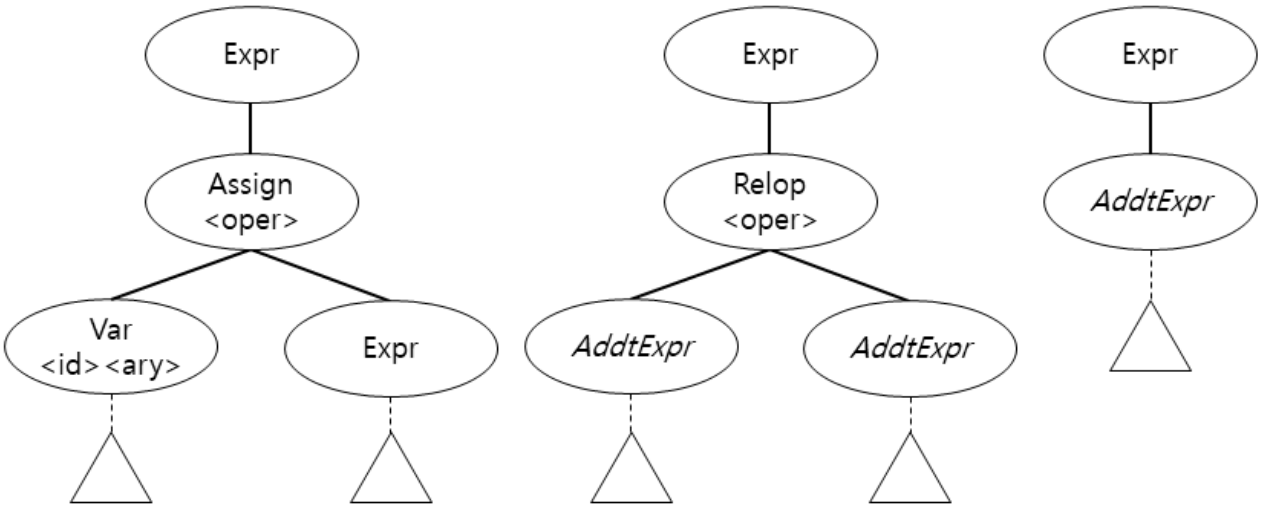
- RetStmt는 Expr를 자식으로 가질 수 있습니다. 자식이 없을 경우 bool empty는 true값이 됩니다.

2.8. Expr

문법

<expression>	::=	<var> ASSIGN <expression>
	|	<simple-expression>
<var>	::=	ID [LSBRACK <expression> RSBRACK]
<simple-expression>	::=	<additive-expression> [<relop> <additive-expression>]

다이어그램



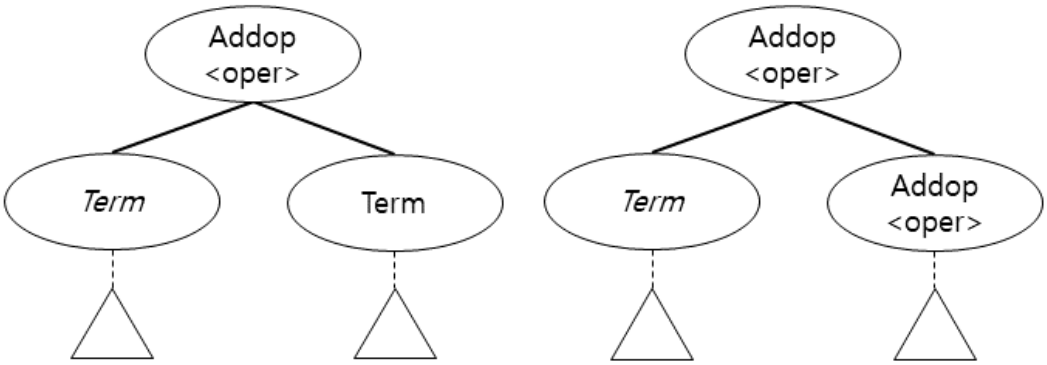
- 현재 문법 수준에서는 위의 세 가지 경우가 가능합니다.
- Assign, Relop, Addop, Mulop 노드는 모두 한가지 Oper 노드 자료구조를 가지기 때문에 이것을 enum oper로 구별합니다.
- AddtExpr 노드는 실제로 존재하지 않습니다. Addop, Mulop, Var, Call, Num, Expr로 대체될 수 있습니다.

2.9. Addop

문법

<additive-expression> ::= <term > { <addop> <term> }

다이어그램



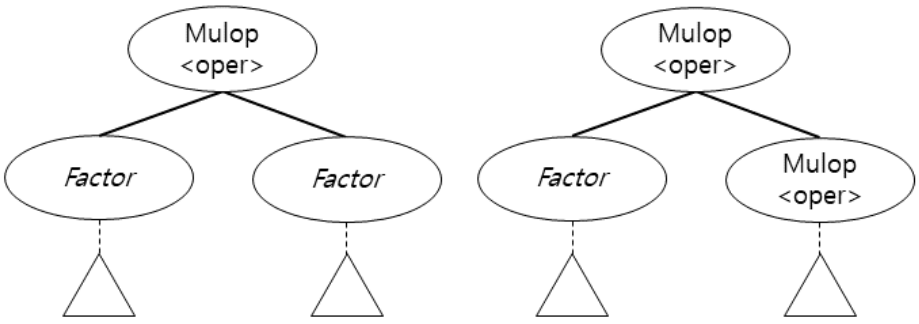
- Addop는 두 개의 자식 노드를 갖습니다. 첫번째 자식은 반드시 Term이고, 두번째 자식은 Addop 또는 Term이 올 수 있습니다.
- Term 노드는 실제로 존재하지 않습니다. Mulop, Var, Call, Num, Expr로 대체될 수 있습니다.

2.10. Mulop

문법

<term> ::= <factor> { <mulop> <factor>}

다이어그램



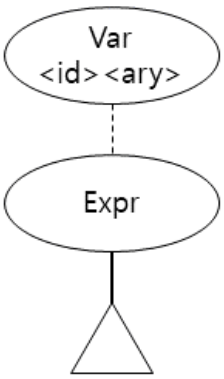
- Mulop는 두 개의 자식 노드를 갖습니다. 첫번째 자식은 반드시 Factor이고, 두번째 자식은 Mulop 또는 Factor가 올 수 있습니다.
- Factor 노드는 실제로 존재하지 않습니다. Var, Call, Num, Expr로 대체될 수 있습니다.

2.11. Var

문법

<var>	::=	ID [LSBRACK <expression> RSBRACK]

다이어그램



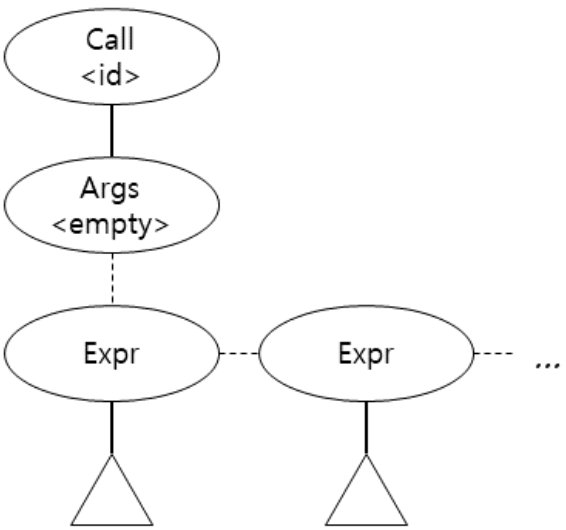
- Var는 Expression으로서만 존재합니다. string id에 이름을 저장하고 배열 표현이 있을 경우 bool ary의 값이 true이며 자식으로 Expr를 갖습니다.

2.12. Call, Args

문법

<call>	::=	ID LPAREN <args> RPAREN
<args>	::=	<arg-list>
	|	EMPTY
<arg-list>	::=	<expression> { COMMA <expression> }

다이어그램



- Call는 string id를 attribute로 가집니다. 자식 노드로 Args를 갖습니다.

- Args는 자식이 없을 경우 bool empty의 값이 true가 됩니다. 만약 있을 경우 Expr를 자식으로 가지며 sibling으로 연결될 수 있습니다.

3. 2.c 실행 결과

```
VarDecl : [x] [true]
| Type : [Int]
| Num : [10]
FunDecl : [minloc] [false]
| Type : [Int]
| ParamList : [false]
| | Param : [a] [true]
| | | Type : [Int]
| | Param : [low] [false]
| | | Type : [Int]
| | Param : [high] [false]
| | | Type : [Int]
| CmpdStmt :
| | LocDecl : [false]
| | | VarDecl : [i] [false]
| | | | Type : [Int]
| | | VarDecl : [x] [false]
| | | | Type : [Int]
| | | VarDecl : [k] [false]
| | | | Type : [Int]
| | StmtList : [false]
| | | ExprStmt : [false]
| | | | Expr :
| | | | | Assign : [Assign]
| | | | | | Var : [k] [false]
| | | | | | Expr :
| | | | | | | Var : [low] [false]
| | | ExprStmt : [false]
| | | | Expr :
| | | | | Assign : [Assign]
| | | | | | Var : [x] [false]
| | | | | | Expr :
| | | | | | | Var : [a] [true]
| | | | | | | | Expr :
| | | | | | | | | Var : [low] [false]
| | | ExprStmt : [false]
| | | | Expr :
| | | | | Assign : [Assign]
| | | | | | Var : [i] [false]
| | | | | | Expr :
| | | | | | | Addop : [ADD]
| | | | | | | | Var : [low] [false]
| | | | | | | | Num : [1]
| | | IterStmt :
| | | | Expr :
| | | | | Relop : [LT]
| | | | | | Var : [i] [false]
| | | | | | Var : [high] [false]
| | | | CmpdStmt :
| | | | | LocDecl : [true]
| | | | | StmtList : [false]
| | | | | | SlctStmt : [false]
| | | | | | | Expr :
| | | | | | | | Relop : [LT]
| | | | | | | | | Var : [a] [true]
```

```
| | | | | | | | | Expr :
| | | | | | | | | | Var : [i] [false]
| | | | | | | | | Var : [x] [false]
| | | | | | | CmpdStmt :
| | | | | | | | LocDecl : [true]
| | | | | | | | StmtList : [false]
| | | | | | | | | ExprStmt : [false]
| | | | | | | | | | Expr :
| | | | | | | | | | | Assign : [Assign]
| | | | | | | | | | | | Var : [x] [false]
| | | | | | | | | | | | Expr :
| | | | | | | | | | | | | Var : [a] [true]
| | | | | | | | | | | | | | Expr :
| | | | | | | | | | | | | | | Var : [i] [false]
| | | | | | | | | ExprStmt : [false]
| | | | | | | | | | Expr :
| | | | | | | | | | | Assign : [Assign]
| | | | | | | | | | | | Var : [k] [false]
| | | | | | | | | | | | Expr :
| | | | | | | | | | | | | Var : [i] [false]
| | | | | | ExprStmt : [false]
| | | | | | | Expr :
| | | | | | | | Assign : [Assign]
| | | | | | | | | Var : [i] [false]
| | | | | | | | | Expr :
| | | | | | | | | | Addop : [ADD]
| | | | | | | | | | | Var : [i] [false]
| | | | | | | | | | | Num : [1]
| | | RetStmt : [false]
| | | | Expr :
| | | | | Var : [k] [false]
FunDecl : [sort] [false]
| Type : [void]
| ParamList : [false]
| | Param : [a] [true]
| | | Type : [Int]
| | Param : [low] [false]
| | | Type : [Int]
| | Param : [high] [false]
| | | Type : [Int]
| CmpdStmt :
| | LocDecl : [false]
| | | VarDecl : [i] [false]
| | | | Type : [Int]
| | | VarDecl : [k] [false]
| | | | Type : [Int]
| | StmtList : [false]
| | | ExprStmt : [false]
| | | | Expr :
| | | | | Assign : [Assign]
| | | | | | Var : [i] [false]
| | | | | Expr :
| | | | | | Var : [low] [false]
| | | IterStmt :
| | | | Expr :
| | | | | Relop : [LT]
| | | | | | Var : [i] [false]
| | | | | Addop : [SUB]
```

```
| | | | | | | Var : [high] [false]
| | | | | | | Num : [1]
| | | | CmpdStmt :
| | | | | LocDecl : [false]
| | | | | | VarDecl : [t] [false]
| | | | | | | Type : [Int]
| | | | | StmtList : [false]
| | | | | | ExprStmt : [false]
| | | | | | | Expr :
| | | | | | | | Assign : [Assign]
| | | | | | | | | Var : [k] [false]
| | | | | | | | | Expr :
| | | | | | | | | | Call : [minloc]
| | | | | | | | | | | Args : [false]
| | | | | | | | | | | | Expr :
| | | | | | | | | | | | | Var : [a] [false]
| | | | | | | | | | | | Expr :
| | | | | | | | | | | | | Var : [i] [false]
| | | | | | | | | | | | Expr :
| | | | | | | | | | | | | Var : [high] [false]
| | | | | | | | | | | | Expr :
| | | | | | | | | | | | | Var : [i] [false]
| | | | | | ExprStmt : [false]
| | | | | | | Expr :
| | | | | | | | Assign : [Assign]
| | | | | | | | | Var : [t] [false]
| | | | | | | | | Expr :
| | | | | | | | | | Var : [a] [true]
| | | | | | | | | | | Expr :
| | | | | | | | | | | | Var : [k] [false]
| | | | | | ExprStmt : [false]
| | | | | | | Expr :
| | | | | | | | Assign : [Assign]
| | | | | | | | | Var : [a] [true]
| | | | | | | | | | Expr :
| | | | | | | | | | | Var : [k] [false]
| | | | | | | | | Expr :
| | | | | | | | | | Var : [a] [true]
| | | | | | | | | | | Expr :
| | | | | | | | | | | | Var : [i] [false]
| | | | | | ExprStmt : [false]
| | | | | | | Expr :
| | | | | | | | Assign : [Assign]
| | | | | | | | | Var : [a] [true]
| | | | | | | | | | Expr :
| | | | | | | | | | | Var : [i] [false]
| | | | | | | | | Expr :
| | | | | | | | | | Var : [t] [false]
| | | | | | ExprStmt : [false]
| | | | | | | Expr :
| | | | | | | | Assign : [Assign]
| | | | | | | | | Var : [i] [false]
| | | | | | | | | Expr :
| | | | | | | | | | Addop : [ADD]
| | | | | | | | | | | Var : [i] [false]
| | | | | | | | | | | Num : [1]
FunDecl : [main] [false]
| Type : [void]
```

```
| ParamList : [true]
| CmpdStmt :
| | LocDecl : [false]
| | | VarDecl : [i] [false]
| | | | Type : [Int]
| | StmtList : [false]
| | | ExprStmt : [false]
| | | | Expr :
| | | | | Assign : [Assign]
| | | | | | Var : [i] [false]
| | | | | | Expr :
| | | | | | | Num : [0]
| | | IterStmt :
| | | | Expr :
| | | | | Relop : [LT]
| | | | | | Var : [i] [false]
| | | | | | Num : [10]
| | | | CmpdStmt :
| | | | | LocDecl : [true]
| | | | | StmtList : [false]
| | | | | | ExprStmt : [false]
| | | | | | | Expr :
| | | | | | | | Assign : [Assign]
| | | | | | | | | Var : [x] [true]
| | | | | | | | | | Expr :
| | | | | | | | | | | Var : [i] [false]
| | | | | | | | | | Expr :
| | | | | | | | | | | Call : [input]
| | | | | | | | | | | | Args : [true]
| | | | | | ExprStmt : [false]
| | | | | | | Expr :
| | | | | | | | Assign : [Assign]
| | | | | | | | | Var : [i] [false]
| | | | | | | | | Expr :
| | | | | | | | | | Addop : [ADD]
| | | | | | | | | | | Var : [i] [false]
| | | | | | | | | | | Num : [1]
| | | ExprStmt : [false]
| | | | Expr :
| | | | | Call : [sort]
| | | | | | Args : [false]
| | | | | | | Expr :
| | | | | | | | Var : [x] [false]
| | | | | | | Expr :
| | | | | | | | Num : [0]
| | | | | | | Expr :
| | | | | | | | Num : [10]
| | | ExprStmt : [false]
| | | | Expr :
| | | | | Assign : [Assign]
| | | | | | Var : [i] [false]
| | | | | | Expr :
| | | | | | | Num : [0]
| | | IterStmt :
| | | | Expr :
| | | | | Relop : [LT]
| | | | | | Var : [i] [false]
| | | | | | Num : [10]
```

```
| | | | CmpdStmt :
| | | | | LocDecl : [true]
| | | | | StmtList : [false]
| | | | | | ExprStmt : [false]
| | | | | | | Expr :
| | | | | | | | Call : [output]
| | | | | | | | | Args : [false]
| | | | | | | | | | Expr :
| | | | | | | | | | | Var : [x] [true]
| | | | | | | | | | | | Expr :
| | | | | | | | | | | | | Var : [i] [false]
| | | | | | ExprStmt : [false]
| | | | | | | Expr :
| | | | | | | | Assign : [Assign]
| | | | | | | | | Var : [i] [false]
| | | | | | | | | Expr :
| | | | | | | | | | Addop : [ADD]
| | | | | | | | | | | Var : [i] [false]
| | | | | | | | | | | Num : [1]
```

- 들여쓰기 세로줄 처리는 보고서에만 표시했습니다.

4. 에러 처리

에러 처리 원칙은 다음과 같습니다.

- match(SEMI)에 실패할 경우, 다음 SEMI를 찾을 때까지 반복해서 현재 토큰을 버립니다.
- 괄호('(', ')') 또는 중괄호('{', '}')가 없는 상황 등으로 인해 Decl 노드를 구성하는데 실패할 경우, 현재 토큰을 버립니다.
- 모든 함수는 nullptr를 반환하지 않습니다.

이렇게 할 경우 무한루프를 예방할 수 있습니다.

4.1. Statement 에러 예시

프로그램

```
void main(void) {
    a = 10 // error: SEMI 없음
    b = 20;
}
```

실행결과

```
>>>Syntax error at line 2: unexpected token -> (was: b)
FunDecl : [main] [false]
| Type : [void]
| ParamList : [true]
| CmpdStmt :
| | LocDecl : [true]
| | StmtList : [false]
| | | ExprStmt : [false]
| | | | Expr :
| | | | | Assign : [Assign]
| | | | | | Var : [a] [false]
| | | | | | Expr :
| | | | | | | Num : [10]
| | | ExprStmt : [true]
```

4.2. Declaration 에러 예시

프로그램

```
void fun (void) {
    a = 10;
}

int b;

void main void) { // error: LP 없음
    c = 30;
}
```

실행결과

```
>>>Syntax error at line 7: unexpected token -> (was: void)

>>>Syntax error at line 7: Code ends before file
(was: ))
FunDecl : [fun] [false]
| Type : [void]
| ParamList : [true]
| CmpdStmt :
| | LocDecl : [true]
| | StmtList : [false]
| | | ExprStmt : [false]
| | | | Expr :
| | | | | Assign : [Assign]
| | | | | | Var : [a] [false]
| | | | | | Expr :
| | | | | | | Num : [10]
VarDecl : [b] [false]
| Type : [Int]
Decl : [main] [false]
| Type : [void]
```