zcay 와 문법 차이

21.03.21
김지현

tinyc.g4

```
program
    : statement +
    ;

statement
    : 'if' paren_expr statement
    | 'if' paren_expr statement 'else' statement
    | 'while' paren_expr statement
    | 'do' statement 'while' paren_expr ';'
    | '{' statement* '}'
    | expr ';'
    | ';'
    ;

paren_expr
    : '(' expr ')'
    ;
```

```
expr
    : test
    | id '=' expr
    ;

test
    : sum
    | sum '<' sum
    ;

sum
    : term
    | sum '+' term
    | sum '-' term
    ;

term
    : id
    | integer
    | paren_expr
    ;
```

```
id
    : STRING
    ;

integer
    : INT
    ;


STRING
    : [a-z]+
    ;


INT
    : [0-9] +
    ;

WS
    : [ \r\n\t] -> skip
    ;
```

공통점

1. 전체적으로 선언하는 구조는 동일
2. 변수가 아닌것은 ' ' 안에 표현 ex) 'for' 'while' 등

차이점

1. “=”으로 지정함

‘+=’ 연산도 주로 사용해서 지정함

```
pragma
   : (name=('zkay' | 'solidity') ver=version) # VersionPragma;

enumValue
  : idf=identifier ;

block
  : '{' statements+=statement* '}' ;
```

```
    | 'do' statement 'while' paren_expr ';'
    | '{' statement* '}'
```

차이점

2. WS(공백)을 표현할 때 channel(HIDDEN) 사용함

```
WS
  : [ \t\r\n\u000C]+ -> channel(HIDDEN) ;
```

```
WS
    : [ \r\n\t] -> skip
    ;
```

차이점

3. identifier 선언할때 $_ 사용 -> default 변수를 의미함

```
fragment
IdentifierStart
  : [a-zA-Z$_] ;

fragment
IdentifierPart
  : [a-zA-Z0-9$_] ;

Identifier
  : IdentifierStart IdentifierPart* ;
```

차이점

4. ? 의 의미

condition연산을 의미, 다음과같이 '='을 이용해 지정하고 사용

```
cond=expression '?' then_expr=expression ':' else_expr=expression # IteExpr

ifStatement
  : 'if' '(' condition=expression ')' then_branch=statement ( 'else' else_branch=statement )? ;

forStatement
  : 'for' '(' ( init=simpleStatement | ';' ) condition=expression? ';' update=expression? ')' body=statement ;
```

헷갈리는 부분 ? codition이 아닌데도 '?'를 사용하는 것도 존재함

```
variableDeclarationStatement
  : variable_declaration=variableDeclaration ( '=' expr=expression )? ';';
```

Visitor 코드 구조 설명

-visitor 클래스는 BaseVisitor 를 상속함

```
public class MyVisitor extends tinycBaseVisitor<Object> {
```

-baseVisitor 의 메소드들을 오버라이드 할 수 있음

-visitor 사용 코드

```
tinycLexer lexer = new tinycLexer(CharStreams.fromFileName("src/test.c"));
CommonTokenStream token = new CommonTokenStream(lexer);

tinycParser parser = new tinycParser(token);
ParseTree tree = parser.program();

MyVisitor visitor = new MyVisitor();
visitor.visit(tree);
```

Visitor 코드 구조 설명

-Visitor의 역할 : 이미생성된 ParseTree를 순회하는 역할

-Visit 룰() : 언제나 visitChidlren(ctx) 메서드를 호출함

-문법에 틀려 파싱에 실패한 노드도 호출가능함