컴파일러 개론
-2주차-

학번: 201404377
이름: 진승언

문제해결방법

일종의 ENBF인 MiniC.g4 파일에 규칙을 정의한 거에 출력문을 적어주고 g4파일을 run as에서 Generate Antlr Recognizer를 해주면 규칙에 맞게 컴파일러를 만들어주고 해당 규칙에 맞는 토큰이 오면 내가 정의한 출력문을 출력해준다. N-N2에서 N은 하나의 큰 규칙 N2는

MiniCParser.java | MiniC.g4 | MiniC.tokens | MiniCListener.java | TestMiniC.java

```
1  grammar MiniC;
2
3  program : decl+          {System.out.println("201404377 Rule 0");};
4  decl        : var_decl   {System.out.println("201404377 Rule 1-1");}
5          | fun_decl       {System.out.println("201404377 Rule 1-2");};
6  var_decl    :  type_spec IDENT ';' {System.out.println("201404377 Rule 2-1");}
7          | type_spec IDENT '=' LITERAL ';'   {System.out.println("201404377 Rule 2-2");}
8          | type_spec IDENT '[' LITERAL ']' ';'   {System.out.println("201404377 Rule 2-3");};
9  type_spec   : VOID       {System.out.println("201404377 Rule 3-1");}
10         | INT              {System.out.println("201404377 Rule 3-2");};
11 fun_decl    : type_spec IDENT '(' params ')' compound_stmt {System.out.println("201404377 Rule 4");};
12 params      : param (',' param)*        {System.out.println("201404377 Rule 5-1");}
13         | VOID             {System.out.println("201404377 Rule 5-2");}
14         |                  {System.out.println("201404377 Rule 5-3");};
15 param       : type_spec IDENT       {System.out.println("201404377 Rule 6-1");}
16         | type_spec IDENT '[' ']'   {System.out.println("201404377 Rule 6-2");};
17 stmt        : expr_stmt          {System.out.println("201404377 Rule 7-1");}
18         | compound_stmt          {System.out.println("201404377 Rule 7-2");}
19         | if_stmt           {System.out.println("201404377 Rule 7-3");}
20         | while_stmt             {System.out.println("201404377 Rule 7-4");}
21         | return_stmt            {System.out.println("201404377 Rule 7-5");};
22 expr_stmt   : expr ';'            {System.out.println("201404377 Rule 8");};
23 while_stmt  : WHILE '(' expr ')' stmt   {System.out.println("201404377 Rule 9");};
24 compound_stmt: '{' local_decl* stmt* '}'    {System.out.println("201404377 Rule 10");};
25 local_decl  : type_spec IDENT ';'{System.out.println("201404377 Rule 11-1");}
26         | type_spec IDENT '=' LITERAL ';'   {System.out.println("201404377 Rule 11-2");}
```

그 안의 또 경우의 수라고(규칙) 볼 수 있다.

각 규칙을 차례대로 설명하되 설명한거는 생략하면서 진행하도록 하겠습니다.

라인3(program): 프로그램의 진입점이다. decl이 한번이상 반복된다.

라인4(decl) : var_decl이나 fun_decl이 온다.

라인6(var_decl) : tpye_spec는 VOID나 INT가 오고 IDENT는 다음과 같이 문자로 시작하고 나머지는(뒤) 숫자나 문자로 이루어진다.

라인6(var_decl) : tpye_spec는 VOID나 INT가 오고 IDENT는 다음과 같이 문자로 시작하고 나머지는(뒤) 숫자나 문자로 이루어진다.

```
74 IDENT   : [a-zA-Z_]
75           (   [a-zA-Z_]
76           |  [0-9]
77           )*;
```

```
80 LITERAL:   DecimalConstant    |   OctalConstant    |   HexadecimalConstant    ;
81
82
83 DecimalConstant
84     :   '0'
85     |   [1-9] [0-9]*
86     ;
87
88 OctalConstant
89     :   '0'[0-7]*
90     ;
91
92 HexadecimalConstant
93     :   '0' [xX] [0-9a-fA-F] +
94     ;
95
```

LITERAL은 위와 같이 10진수 9진수 16진수가 올 수 있다.

즉 var_decl은 해석하면 변수선언부라고 볼 수 있다.

라인9(type_spec) : VOID 나 INT가 올 수 있다.

라인11(fun_decl) : 함수 선언이라 볼 수 있다.

```
type_spec IDENT '(' params ')' compound_stmt
```

다음과 같이 타입 문자열 (매개변수)와 복합상태문이 올 수 있다. 뒤에서도 나오겠지만 미리 설명하자면 params는 다음과 같이 하나가 온다면 param하나가 오고 뒤이어 , param이 여러개가 올 수 있다. param은 타입 변수명 , 타입 변수명 [] 이 올 수 있다. compound_stmt는 뒤에서 설명한다.

```
params          : param (',' param)*
{System.out.println("201404377 Rule 5-1");}
                | VOID
{System.out.println("201404377 Rule 5-2");}
                |
{System.out.println("201404377 Rule 5-3");};
param           : type_spec IDENT
{System.out.println("201404377 Rule 6-1");}
                | type_spec IDENT '[' ']'
{System.out.println("201404377 Rule 6-2");};
```

라인12(params) : 1. param 기본으로 하나가 오고 여러개가 올 수 도있는데 두 개 이상일 때는 ,(콤마)가 붙는다.
2. VOID가 올 수 있다.
3. 공백이 올 수 있다.

라인15(param) : 타입 IDENT 나 타입 IDENT[]가 올 수 있다.

라인17(stmt) :

```
stmt        : expr_stmt
{System.out.println("201404377 Rule 7-1");}
            | compound_stmt
{System.out.println("201404377 Rule 7-2");}
            | if_stmt
{System.out.println("201404377 Rule 7-3");}
            | while_stmt
{System.out.println("201404377 Rule 7-4");}
            | return_stmt
{System.out.println("201404377 Rule 7-5");};
```

위와 같이 여러 종류의 stmt중 하나가 올 수 있는데 뒤에서 설명하도록 한다.

라인22(expr_stmt) : expr이 오고 뒤에 ;가 온다.

라인23(while_stmt) :WHILE 이 오고 ( ) 괄호가 오고 괄호안에 expr이온다. 그리고 마지막으로 stmt가 온다.

라인24(compound_stmt) : 중괄호가({ }) 오고 local_decl이 1개 이상오고 stmt가 1개 이상올 수 있다.

라인25(local_decl) :

```
local_decl  : type_spec IDENT
';'{System.out.println("201404377 Rule 11-1");}
            | type_spec IDENT '=' LITERAL ';'
{System.out.println("201404377 Rule 11-2");}
            | type_spec IDENT '[' LITERAL ']' ';'
{System.out.println("201404377 Rule 11-3");};
```

위와 같이 타입 IDENT가 올 수 있다. 즉 IDENT를 정의하거나 값을 바로 할당할 수 있다. 배열도 가능하다.

```
27        | type_spec IDENT '[' LITERAL ']' ';'   {System.out.println("201404377 Rule 11-3");};
28  if_stmt     : IF '(' expr ')' stmt      {System.out.println("201404377 Rule 12-1");}
29        | IF '(' expr ')' stmt ELSE stmt        {System.out.println("201404377 Rule 12-2");};
30  return_stmt : RETURN ';'            {System.out.println("201404377 Rule 13-1");}
31        | RETURN expr ';'                 {System.out.println("201404377 Rule 13-2");};
32  expr    :  LITERAL              {System.out.println("201404377 Rule 14-1");}
33      | '(' expr ')'                {System.out.println("201404377 Rule 14-2");}
34      | IDENT               {System.out.println("201404377 Rule 14-3");}
35      | IDENT '[' expr ']'              {System.out.println("201404377 Rule 14-4");}
36      | IDENT '(' args ')'              {System.out.println("201404377 Rule 14-5");}
37      | '-' expr                {System.out.println("201404377 Rule 14-6");}
38      | '+' expr                {System.out.println("201404377 Rule 14-7");}
39      | '--' expr               {System.out.println("201404377 Rule 14-8");}
40      | '++' expr               {System.out.println("201404377 Rule 14-9");}
41      | expr '*' expr               {System.out.println("201404377 Rule 14-10");}
42      | expr '/' expr               {System.out.println("201404377 Rule 14-11");}
43      | expr '%' expr               {System.out.println("201404377 Rule 14-12");}
44      | expr '+' expr               {System.out.println("201404377 Rule 14-13");}
45      | expr '-' expr               {System.out.println("201404377 Rule 14-14");}
46      | expr EQ expr              {System.out.println("201404377 Rule 14-15");}
47      | expr NE expr                {System.out.println("201404377 Rule 14-16");}
48      | expr LE expr                {System.out.println("201404377 Rule 14-17");}
49      | expr '<' expr               {System.out.println("201404377 Rule 14-18");}
50      | expr GE expr                {System.out.println("201404377 Rule 14-19");}
51      | expr '>' expr               {System.out.println("201404377 Rule 14-20");}
52      | '!' expr                    {System.out.println("201404377 Rule 14-21");}
```

라인28(if_stmt) :

```
if_stmt             : IF '(' expr ')' stmt
{System.out.println("201404377 Rule 12-1");}
            | IF '(' expr ')' stmt ELSE stmt
{System.out.println("201404377 Rule 12-2");};
```

라인32(expr) : 말 그대로 if문 관련이다. expr은

```
expr  :  LITERAL
{System.out.println("201404377 Rule 14-1");}
      | '(' expr ')'
{System.out.println("201404377 Rule 14-2");}
      | IDENT
```

```
{System.out.println("201404377 Rule 14-3");}
        | IDENT '[' expr ']'
{System.out.println("201404377 Rule 14-4");}
        | IDENT '(' args ')'
{System.out.println("201404377 Rule 14-5");}
        | '-' expr
{System.out.println("201404377 Rule 14-6");}
        | '+' expr
{System.out.println("201404377 Rule 14-7");}
        | '--' expr
{System.out.println("201404377 Rule 14-8");}
        | '++' expr
{System.out.println("201404377 Rule 14-9");}
        | expr '*' expr
{System.out.println("201404377 Rule 14-10");}
        | expr '/' expr
{System.out.println("201404377 Rule 14-11");}
        | expr '%' expr
{System.out.println("201404377 Rule 14-12");}
        | expr '+' expr
{System.out.println("201404377 Rule 14-13");}
        | expr '-' expr
{System.out.println("201404377 Rule 14-14");}
        | expr EQ expr
{System.out.println("201404377 Rule 14-15");}
        | expr NE expr
{System.out.println("201404377 Rule 14-16");}
        | expr LE expr
{System.out.println("201404377 Rule 14-17");}
        | expr '<' expr
{System.out.println("201404377 Rule 14-18");}
        | expr GE expr
{System.out.println("201404377 Rule 14-19");}
        | expr '>' expr
{System.out.println("201404377 Rule 14-20");}
        | '!' expr
{System.out.println("201404377 Rule 14-21");}
        | expr AND expr
{System.out.println("201404377 Rule 14-22");}
        | expr OR expr
{System.out.println("201404377 Rule 14-23");}
        | IDENT '=' expr
{System.out.println("201404377 Rule 14-24");}
        | IDENT '[' expr ']' '=' expr
{System.out.println("201404377 Rule 14-25");};
```

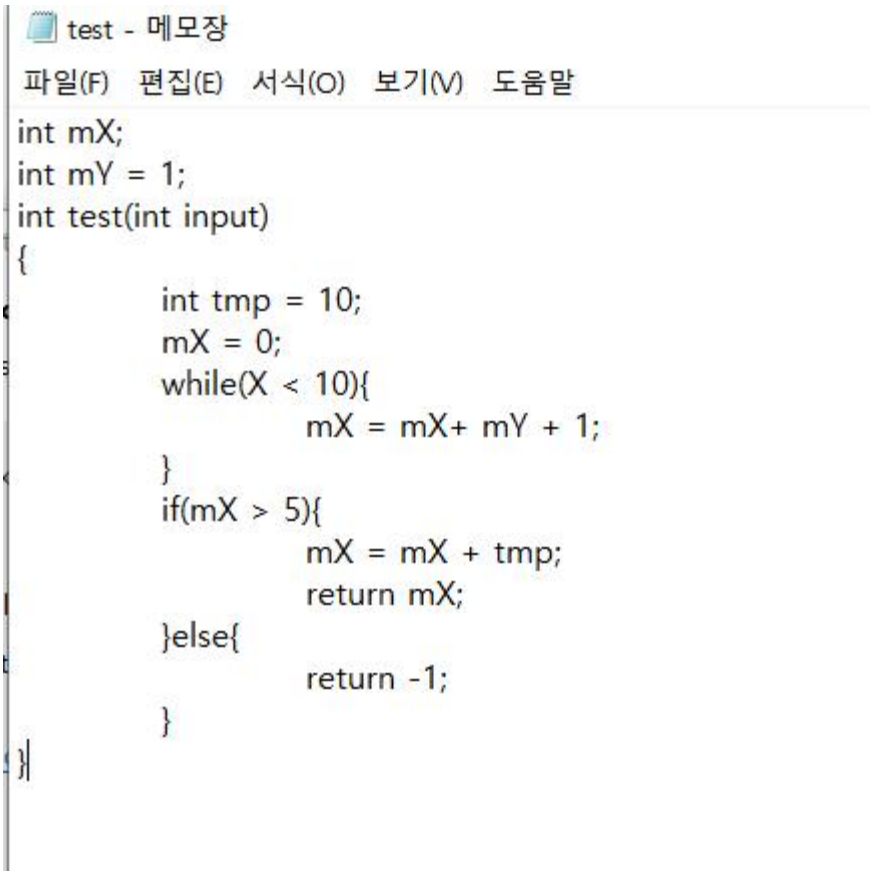
다음과 같이 가장 많은 경우가 있다.

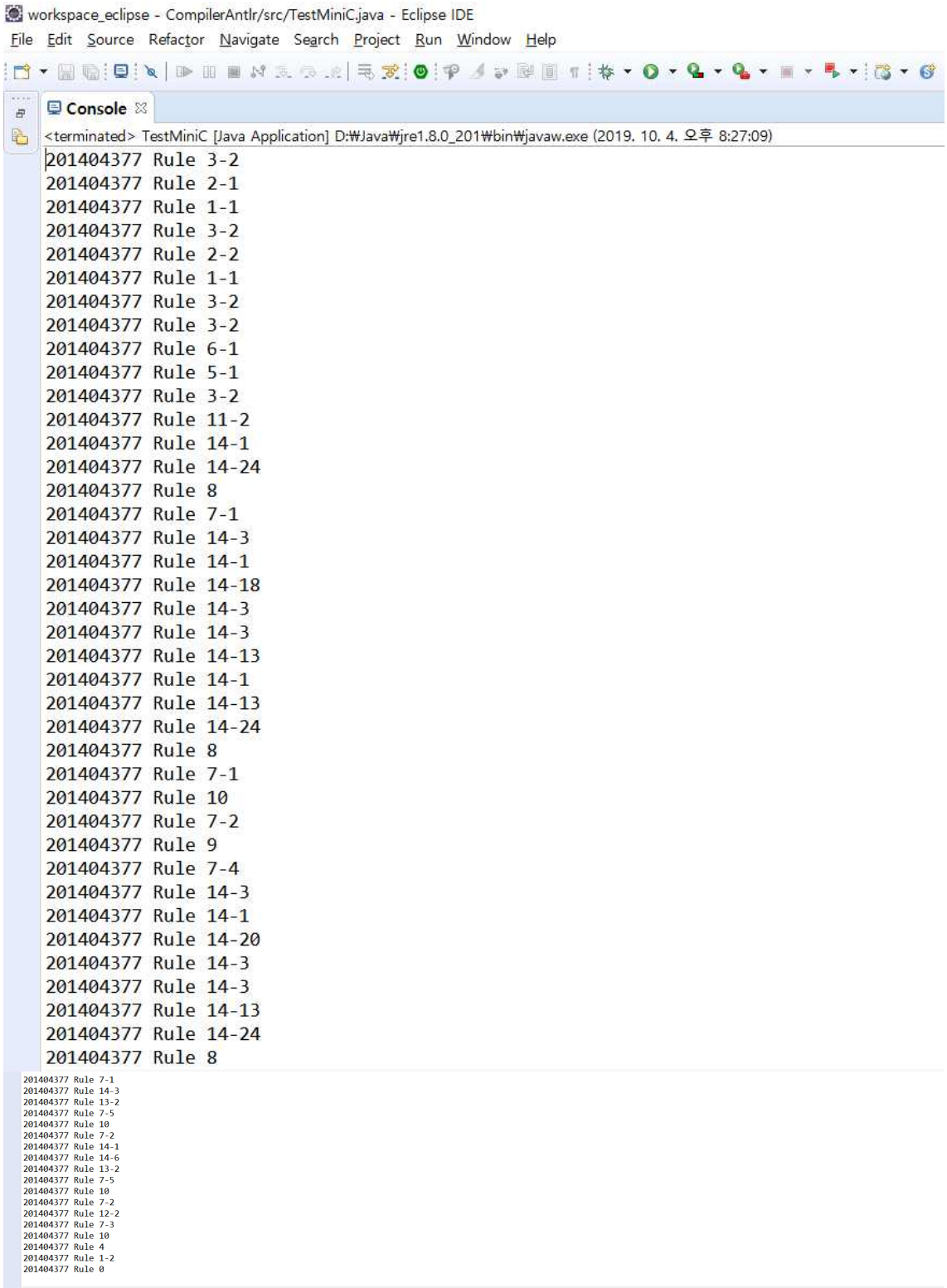
라인57(args) :

```
args  : expr (',' expr)*
{System.out.println("201404377 Rule 15-1");}
        |
{System.out.println("201404377 Rule 15-2");};
```

expr이 한 개 이상올 수 있되 두 개 부터는 앞에 콤마가 붙는다. 또는 공백이 올 수 있다.

테스트 코드 실행 결과


test - 메모장
파일(F) 편집(E) 서식(O) 보기(V) 도움말
int mX;
int mY = 1;
int test(int input)
{
int tmp = 10;
mX = 0;
while(X < 10){
mX = mX+ mY + 1;
}
if(mX > 5){
mX = mX + tmp;
return mX;
}else{
return -1;
}
}


workspace_eclipse - CompilerAntlr/src/TestMiniC.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
Console
<terminated> TestMiniC [Java Application] D:₩Java₩jre1.8.0_201₩bin₩javaw.exe (2019. 10. 4. 오후 8:27:09)
201404377 Rule 3-2
201404377 Rule 2-1
201404377 Rule 1-1
201404377 Rule 3-2
201404377 Rule 2-2
201404377 Rule 1-1
201404377 Rule 3-2
201404377 Rule 3-2
201404377 Rule 6-1
201404377 Rule 5-1
201404377 Rule 3-2
201404377 Rule 11-2
201404377 Rule 14-1
201404377 Rule 14-24
201404377 Rule 8
201404377 Rule 7-1
201404377 Rule 14-3
201404377 Rule 14-1
201404377 Rule 14-18
201404377 Rule 14-3
201404377 Rule 14-3
201404377 Rule 14-13
201404377 Rule 14-1
201404377 Rule 14-13
201404377 Rule 14-24
201404377 Rule 8
201404377 Rule 7-1
201404377 Rule 10
201404377 Rule 7-2
201404377 Rule 9
201404377 Rule 7-4
201404377 Rule 14-3
201404377 Rule 14-1
201404377 Rule 14-20
201404377 Rule 14-3
201404377 Rule 14-3
201404377 Rule 14-13
201404377 Rule 14-24
201404377 Rule 8
201404377 Rule 7-1
201404377 Rule 14-3
201404377 Rule 13-2
201404377 Rule 7-5
201404377 Rule 10
201404377 Rule 7-2
201404377 Rule 14-1
201404377 Rule 14-6
201404377 Rule 13-2
201404377 Rule 7-5
201404377 Rule 10
201404377 Rule 7-2
201404377 Rule 12-2
201404377 Rule 7-3
201404377 Rule 10
201404377 Rule 4
201404377 Rule 1-2
201404377 Rule 0

총 과제 수행에 걸린 시간

해결하는데 2시간 + 보고서 쓰는데 2시간 = 4시간