

컴파일러 개론

-3주차-

분반: 00

학번: 201404377

이름: 진승언

[문제 해결방법]



```
1 |
2+ import org.antlr.v4.runtime.ANTLRFilestream;
10
11 public class TestMiniC {
12+ public static void main(String[] args) throws Exception {
13     MiniCLexer lexer = new MiniCLexer( new ANTLRFilestream("test.c"));
14     CommonTokenStream tokens = new CommonTokenStream( lexer );
15     MiniCParser parser = new MiniCParser( tokens );
16     ParseTree tree = parser.program();
17
18     ParseTreeWalker walker = new ParseTreeWalker();
19     walker.walk(new MiniCPrintListener(), tree);
20
21 }
22 }
23
```

ANTLR의 walker가 walk을 하면 파스트리의 각 노드를 depth-first로 방문하는데, preorder 로 수행하고 싶은 것은 enterXXX 에, postorder 로 수행하고 싶은 것은 exitXXX 에 들어있게 된다. 나는 exitXXX로 구현하였다.

자식노드에서는 put()으로 자신의 ctx 에 새로운 text 를 저장하고 부모노드에서는 children 의 ctx'에 대해 get(ctx')로 읽어올 수 있다.

즉, 모든 토큰들을 newTexts에(key, value 구조) 저장하고 ctx는 각 문법에 맞는 토큰들이 있다고 생각하면 된다. 이 ctx들을 newTexts에 저장하면서 후위순회를 하고 최종적으로 문자열의 줄 바꿈 간격이 맞게 조정해주면 과제가 해결된다. 간단히 말하면 newTexts는 모든 토큰들이 저장될 공간, 즉 트리 전체이고 ctx는 자신까지 오기 전인(후위 순회 하기 전인) 하위노드들이 저장되어 있다고 보면 된다.

이쁘게 출력해줘야 하므로 알맞게 찍어쓰기 줄바꿈도 해주어야한다.

```

MiniCPrintListener.java  MiniC.g4  TestMiniC.java
1 // 문법 이름이 MiniC 이다.
2 grammar MiniC;
3
4 // 나중에 ANTLR4 가 어휘분석, 구문분석을 하는 .java 파일들을 생성할 때,
5 // generated 라는 package 에 생성하게 된다.
6 @header {
7
8
9
10
11 program : decl+ {System.out.println("201404377 Rule 0");};
12 decl : var_decl {System.out.println("201404377 Rule 1-1");};
13 | fun_decl {System.out.println("201404377 Rule 1-2");};
14 var_decl : type_spec IDENT ';' {System.out.println("201404377 Rule 2-1");};
15 | type_spec IDENT '=' LITERAL ';' {System.out.println("201404377 Rule 2-2");};
16 | type_spec IDENT '[' LITERAL ']' ';' {System.out.println("201404377 Rule 2-3");};
17 type_spec : VOID {System.out.println("201404377 Rule 3-1");};
18 | INT {System.out.println("201404377 Rule 3-2");};
19 fun_decl : type_spec IDENT '(' params ')' compound_stmt {System.out.println("201404377 Rule 4");};
20 params : param (',' param)* {System.out.println("201404377 Rule 5-1");};
21 | VOID {System.out.println("201404377 Rule 5-2");};
22 | {System.out.println("201404377 Rule 5-3");};
23 param : type_spec IDENT {System.out.println("201404377 Rule 6-1");};
24 | type_spec IDENT '[' ']' {System.out.println("201404377 Rule 6-2");};
25 stmt : expr_stmt {System.out.println("201404377 Rule 7-1");};
26 | compound_stmt {System.out.println("201404377 Rule 7-2");};
27 | if_stmt {System.out.println("201404377 Rule 7-3");};
28 | while_stmt {System.out.println("201404377 Rule 7-4");};
29 | return_stmt {System.out.println("201404377 Rule 7-5");};
30

```

먼저 난 MiniC.g4의(MinicBaseListener.java) 순서대로 과제를 진행했다.
program만 가장 마지막으로 했다.

해결방법을 차례대로 알아 보도록 하면 다음과 같다. 참고로 나는 조교님의 말씀대로 보조 메소드를 만들어 적극적으로 사용하였다. 주석에는 문법을 적어놨다.

1. decl

```
42- /*
43  * decl : var_decl | fun_decl
44  */
45- @Override
46  public void exitDecl(MiniCParser.DeclContext ctx) {
47
48      if (isVarDecl(ctx)) {
49          String var_decl = newTexts.get(ctx.var_decl());
50          newTexts.put(ctx, var_decl);
51      } else {
52          String fun_decl = newTexts.get(ctx.fun_decl());
53          newTexts.put(ctx, fun_decl);
54      }
55
56  }
```

var_decl 혹은 fun_decl이 올 수 있다. 선언문이라 보면 된다.

```
332  //var_decl 인지
333- boolean isVarDecl(MiniCParser.DeclContext ctx) {
334      return ctx.var_decl() == ctx.getChild(0);
335  }
```

var_decl인지 위와 같이 ctx의 첫 번째 child가 var_decl 타입인지 확인하는 메소드를 만들어 사용하였다. 이렇게 확인 후 var_decl 또는 fun_decl을 newTexts에 put해주면 된다. 이 이후로 모든 exitXXX를 모두 이런 식으로 해결한다. |, +, * (정규식)에 따라 분기처리만 해주고 해당 문법을 newTexts에 넣어주면 된다.

2. var_decl

```
58- /*
59  * var_decl : type_spec IDENT ';' | type_spec IDENT '=' LITERAL ';' | type_spec
60  * IDENT '[' LITERAL ']' ';'
61  */
62- @Override
63  public void exitVar_decl(MiniCParser.Var_declContext ctx) {
64
65      String type_spec = newTexts.get(ctx.type_spec());
66      String ident = ctx.getChild(1).getText();
67
68      if (isVarDeclOne(ctx)) {
69          newTexts.put(ctx, type_spec + " " + ident + ";");
70      } else if (isVarDeclDefineValue(ctx)) {
71          String literal = ctx.getChild(3).getText();
72          newTexts.put(ctx, type_spec + " " + ident + " = " + literal + ";");
73      } else { // 배열 String literal =
74          String literal = ctx.getChild(3).getText();
75          newTexts.put(ctx, type_spec + " " + ident + " [" + literal + "]" + ";");
76      }
77  }
```

변수 선언이다. 주석에 써있는 var_decl 문법을 차례대로 해결했다. 먼저 isVarDeclOne()는 첫 번째의 단순 변수 선언인지 확인하는 메소드이다.

isVarDeclDefineValue()는 변수선언과 동시에 초기화하는지 확인하는 메소드이다. 마지막 else문은 배열을 담당한다. 이렇게 분기 처리 후 각 문법에 맞게 newTexts에 추가해주면 된다.

```

82 IDENT : [a-zA-Z_]
83       ( [a-zA-Z_]
84         | [0-9]
85       )*;
86
87

```

literal는 위와 같다. 즉 변수 명명규칙에 따른 ID 와 숫자라고 보면된다. 이러한 terminal node들은 getChild(position)으로 가져와 getText()후 저장하면 된다.

3. type_spec

```

80 /*
81  * type_spec : VOID | INT ;
82  */
83 @Override
84 public void exitType_spec(MiniCParser.Type_specContext ctx) {
85     String type = ctx.getChild(0).getText();
86     newTexts.put(ctx, type);
87 }
88

```

타입이다. type_spec는 VOID 나 INT가 올 수 있다. 즉 타입이다. VOID와 INT도 terminal node이므로 getChild().getText()로 가져와 newTexts에 저장해주면 된다.

4. fun_decl

```

89 /*
90  * fun_decl : type_spec IDENT '(' params ')' compound_stmt ;
91  */
92 @Override
93 public void exitFun_decl(MiniCParser.Fun_declContext ctx) {
94
95     String type_spec = newTexts.get(ctx.type_spec());
96     String ident = ctx.getChild(1).getText();
97     String params = newTexts.get(ctx.params());
98     String compound_stmt = newTexts.get(ctx.compound_stmt());
99     newTexts.put(ctx, type_spec + " " + ident + "(" + params + ")\n" + compound_stmt);
100 }
101

```

함수 선언이다. 지금까지 했던 것처럼 각각의 ctx에 맞게 뽑아낸 후 newTexts에 저장해주면 된다. 함수 선언 후 줄바꿈을 해야하므로 “\n”을 추가해준다.

5. params

```
102-  /*
103-   * params : param (',' param)* | VOID | ;
104-   */
105-  @Override
106-  public void exitParams(MiniCParser.ParamsContext ctx) {
107-
108-      if (isParamsMultiple(ctx)) { // param이 여러개일 경우
109-          String param = newTexts.get(ctx.param(0));
110-          for (int i = 1; i < ctx.param().size(); i++) {
111-              param = param + "," + newTexts.get(ctx.param(i));
112-          }
113-          newTexts.put(ctx, param);
114-      } else if (isParamsOne(ctx)) { // param이 하나인 경우
115-          String param = newTexts.get(ctx.param(0));
116-          newTexts.put(ctx, param);
117-      } else if (isParamsVoid(ctx)) { // void
118-          String VOID = ctx.getChild(0).getText();
119-          newTexts.put(ctx, VOID);
120-      } else { // EMPTY
121-          String empty = "";
122-          newTexts.put(ctx, empty);
123-      }
124-  }
```

파라미터들 이다. 여러개가 올 수 도 있고 없을 수도 있고 VOID가 올 수 도 있다.

```
347-  // param이 여러개인지(매개변수가 여러개 올 경우)
348-  private boolean isParamsMultiple(MiniCParser.ParamsContext ctx) {
349-      return ctx.getChildCount() > 1;
350-  }
351-
352-  // param이 한개인지
353-  private boolean isParamsOne(MiniCParser.ParamsContext ctx) {
354-      return ctx.getChildCount() == 1 && ctx.param() != null;
355-  }
356-
357-  // param이 void 인지
358-  private boolean isParamsVoid(MiniCParser.ParamsContext ctx) {
359-      return ctx.getChildCount() == 1 && ctx.VOID() != null;
360-  }
361-
```

isParamsMultiple 메소드를 통해 param이 여러개인지 확인하고 여러개라면 있는 개수만큼 반복문을 돌린 후 저장한다. 다음 분기문은 param이 하나라면 해당 param을 저장한다. 그리고 void인 경우는 void는 terminal node이므로 getChild로 가져와 저장해주면 된다. 마지막으로 empty 비어있는 경우는 빈 문자열을 저장해주면 된다.

6. param

```
126- /*
127-  * param : type_spec IDENT | type_spec IDENT '[' ']' ; |
128-  */
129- @Override
130- public void exitParam(MiniCParser.ParamContext ctx) {
131-     String type_spec = newTexts.get(ctx.type_spec());
132-     String ident = ctx.getChild(1).getText();
133-     if (isParamArray(ctx)) { // 배열인 경우
134-         newTexts.put(ctx, type_spec + " " + ident + " " + "[" + "]");
135-     } else {
136-         newTexts.put(ctx, type_spec + " " + ident);
137-     }
138- }
139
```

파라미터이다. 파라미터가 단순 변수이거나 배열인 경우가 있다. 둘 다 type_spec와 IDENT로 되었으며 배열인 경우에만 뒤에 []을 붙여주면 된다.

7. stmt

```
140- /*
141-  * stmt : expr_stmt | compound_stmt | if_stmt | while_stmt | return_stmt ;
142-  */
143- @Override
144- public void exitStmt(MiniCParser.StmtContext ctx) {
145-
146-     if (ctx.getChild(0) == ctx.expr_stmt()) {
147-         String expr_stmt = newTexts.get(ctx.expr_stmt());
148-         newTexts.put(ctx, expr_stmt);
149-     } else if (ctx.getChild(0) == ctx.compound_stmt()) {
150-         String compound_stmt = newTexts.get(ctx.compound_stmt());
151-         newTexts.put(ctx, compound_stmt);
152-     } else if (ctx.getChild(0) == ctx.if_stmt()) {
153-         String if_stmt = newTexts.get(ctx.if_stmt());
154-         newTexts.put(ctx, if_stmt);
155-     } else if (ctx.getChild(0) == ctx.while_stmt()) {
156-         String while_stmt = newTexts.get(ctx.while_stmt());
157-         newTexts.put(ctx, while_stmt);
158-     } else if (ctx.getChild(0) == ctx.return_stmt()) {
159-         String return_stmt = newTexts.get(ctx.return_stmt());
160-         newTexts.put(ctx, return_stmt);
161-     }
162- }
```

상태문 이다. 총 5가지가 있다. 각 타입에 맞게 분기문 처리하고 가져와서 저장해주면 된다.

8. expr_stmt

```
164  /*
165   * expr_stmt : expr ';' ;
166   */
167  @Override
168  public void exitExpr_stmt(MiniCParser.Expr_stmtContext ctx) {
169      String expr = newTexts.get(ctx.expr());
170      newTexts.put(ctx, expr + ";");
171  }
```

표현식 상태문이다. expr를 가져온 후 저장해주면되는데 뒤에 ;를 붙여줘야한다.

9. while_stmt

```
173  /*
174   * while_stmt : WHILE '(' expr ')' stmt ;
175   */
176  @Override
177  public void exitWhile_stmt(MiniCParser.While_stmtContext ctx) {
178
179      String WHILE = ctx.getChild(0).getText();
180      String expr = newTexts.get(ctx.expr());
181      String stmt = newTexts.get(ctx.stmt());
182      newTexts.put(ctx, WHILE + " " + "(" + expr + ")" + "\n" + stmt);
183  }
```

while문 이다. 경우의 수가 한가지이므로 똑같이 각각에 맞게 뽑아온 후 저장 해주면 된다. WHILE도 terminal node이므로 getChild().getText()로 뽑아와 저장하고 while() 문 다음은 한줄일 띄워주기 위해 “\n”을 추가해준다.

10. comound_stmt

```
185  /*
186   * compound_stmt: '{' local_decl* stmt* '}' ; // * 0개 이상
187   */
188  @Override
189  public void exitCompound_stmt(MiniCParser.Compound_stmtContext ctx) {
190
191      String local_decl = "";
192      String stmt = "";
193
194      for (int i = 0; i < ctx.local_decl().size(); i++) {
195          local_decl = local_decl + newTexts.get(ctx.local_decl(i)) + "\n";
196      }
197
198      for (int i = 0; i < ctx.stmt().size(); i++) {
199          stmt = stmt + newTexts.get(ctx.stmt(i)) + "\n";
200      }
201      newTexts.put(ctx, "{\n" + local_decl + stmt + "}");
202
203  }
```

복합상태문이다. 처음과 끝에 { } 가 오고 local_decl이 0개 이상 stmt도 뒤이

어 0개 이상 올 수 있다. 0개 이상 올 수 있으므로 각각 for문으로 받아온 후 저장해준다. for문으로 저장해줄 때 줄바꿈도 해준다. 저장할 때 처음과 끝에 }를 붙여주고 { 이후에는 줄바꿈을 해준다. } 전에는 local_decl과 stmt 끝에 줄바꿈이 붙으므로 해줄 필요가 없다.

11. local_decl

```

205-  /*
206   * local_decl : type_spec IDENT ';' | type_spec IDENT '=' LITERAL ';' |
207   * type_spec IDENT '[' LITERAL ']' ';' ;
208   */
209-  @Override
210  public void exitLocal_decl(MiniCParser.Local_declContext ctx) {
211
212      String type_spec = newTexts.get(ctx.type_spec());
213      String ident = ctx.getChild(1).getText();
214      if (isLocalDeclOne(ctx)) {
215          newTexts.put(ctx, type_spec + " " + ident + ";");
216      } else if (isLocalDeclDefineValue(ctx)) {
217          String literal = ctx.getChild(2).getText();
218          newTexts.put(ctx, type_spec + " " + ident + " " + "=" + " " + literal + ";");
219      } else {
220          String literal = ctx.getChild(3).getText();
221          newTexts.put(ctx, type_spec + " " + ident + " " + "[" + literal + "]" + ";");
222      }
223  }
224

```

지역 변수 선언이다. 2번 var_decl과 방식은 동일하다.

12. if_stmt

```

225-  /*
226   * if_stmt : IF '(' expr ')' stmt | IF '(' expr ')' stmt ELSE stmt ;
227   */
228-  @Override
229  public void exitIf_stmt(MiniCParser.If_stmtContext ctx) {
230
231      String IF = ctx.getChild(0).getText();
232      String expr = newTexts.get(ctx.expr());
233      String stmt = newTexts.get(ctx.stmt(0));
234
235      if (newTexts.get(ctx.stmt(0).compound_stmt()) == null) {
236          stmt = "{\n" + stmt + "\n}";
237      }
238      if (isIfStmtOnlyIf(ctx)) {
239          newTexts.put(ctx, IF + "(" + expr + ")\n" + stmt);
240      } else {
241          String ELSE = ctx.getChild(5).getText();
242          String stmt2 = newTexts.get(ctx.stmt(1));
243          newTexts.put(ctx, IF + "(" + expr + ")\n" + stmt + "\n" + ELSE + "\n" + stmt2);
244      }
245  }
246

```

if문이다. 단일 if문과 if else문이 나올 수 있다. 만약 복합 상태문인 경우 stmt에 라인 235처럼 중괄호와 줄바꿈으로 감싸서 stmt를 저장해 주도록 한다. 그다음 if문만 있는 경우는 if문 문법에 맞게 newTexts에 저장해주도록 하고 if else 문인 경우 그에 맞게 newTexts에 저장해주도록 한다. IF문도 terminal node이므로 getChild().getText()로 가져오도록 한다.

보조 메소드로 사용한 isIfStmtOnlyIf는 다음과 같다.

```
378 // if문만 있는 조건문인지
379 boolean isIfStmtOnlyIf(MiniParser.If_stmtContext ctx) {
380     return ctx.getChildCount() == 5;
381 }
382
```

13. return_stmt

```
247 /*
248  * return_stmt : RETURN ';' | RETURN expr ';' ;
249  */
250 @Override
251 public void exitReturn_stmt(MiniParser.Return_stmtContext ctx) {
252     String RETURN = ctx.getChild(0).getText();
253     if (isReturnStmtHasReturnValue(ctx)) {
254         String expr = newTexts.get(ctx.expr());
255         newTexts.put(ctx, RETURN + " " + expr + ";");
256     } else {
257         newTexts.put(ctx, RETURN + ";");
258     }
259 }
```

return 상태문이다. return; 이나 return expr이(ex. return a; , return 3;) 올 수 있다. return도 terminal node이므로 getChild().getText()로 가져오면 된다. 그 후 두가지 경우를 분기처리하여 알맞게 저장해주면 된다. isReturnStmtHasReturnValue()는 다음과 같다.

```
383 // 리턴할 값이 있는 리턴문 ex) return 1;
384 boolean isReturnStmtHasReturnValue(MiniParser.Return_stmtContext ctx) {
385     return ctx.getChildCount() == 3;
386 }
387
```

14. expr

```

261  /*
262   * expr : LITERAL | '(' expr ')' | IDENT | IDENT '[' expr ']' | IDENT '(' args
263   *        ')' | '-' expr | '+' expr | '--' expr | '++' expr | expr '*' expr | expr '/'
264   *        expr | expr '%' expr | expr '+' expr | expr '-' expr | expr EQ expr | expr NE
265   *        expr | expr LE expr | expr '<' expr | expr GE expr | expr '>' expr | '!' expr
266   *        | expr AND expr | expr OR expr | IDENT '=' expr | IDENT '[' expr ']' '=' expr
267   */
268
269  @Override
270  public void exitExpr(MiniParser.ExprContext ctx) {
271
272      String ident = null, expr = null, expr2 = null, args = null, op = null;
273      if (ctx.getChildCount() == 1) {
274          newTexts.put(ctx, ctx.getChild(0).getText());
275      } else if (ctx.getChildCount() == 3 && ctx.getChild(1) == ctx.expr()) {
276          expr = newTexts.get(ctx.expr(0));
277          newTexts.put(ctx, "(" + expr + ")");
278      } else if (ctx.getChildCount() == 4 && ctx.getChild(0) == ctx.IDENT() && ctx.getChild(2) == ctx.expr()) {
279          expr = newTexts.get(ctx.expr(0));
280          ident = ctx.getChild(0).getText();
281          newTexts.put(ctx, ident + "[" + expr + "]");
282      } else if (ctx.getChildCount() == 4 && ctx.getChild(0) == ctx.IDENT() && ctx.getChild(2) == ctx.args()) {
283
284          args = newTexts.get(ctx.args());
285          ident = ctx.getChild(0).getText();
286          newTexts.put(ctx, ident + "(" + args + ")");
287      } else if (ctx.getChildCount() == 2) {
288          op = ctx.getChild(0).getText();
289          expr = newTexts.get(ctx.expr(0));
290          newTexts.put(ctx, op + expr);
291      } else if (isBinaryOperation(ctx)) { // expr op expr | IDENT op expr
292          if (ctx.getChild(0) == ctx.IDENT()) { //
293              ident = ctx.getChild(0).getText();
294              op = ctx.getChild(1).getText();
295              expr = newTexts.get(ctx.expr(0));
296              newTexts.put(ctx, ident + " " + op + " " + expr);
297          } else { // 첫번째 요소가 expr
298              expr = newTexts.get(ctx.expr(0));
299              expr2 = newTexts.get(ctx.expr(1));
300              op = ctx.getChild(1).getText();
301              newTexts.put(ctx, expr + " " + op + " " + expr2);
302          }
303      } else if (ctx.getChildCount() == 6) {
304          ident = ctx.getChild(0).getText();
305          expr = newTexts.get(ctx.expr(0));
306          expr2 = newTexts.get(ctx.expr(1));
307          newTexts.put(ctx, ident + "[" + expr + "]" + " " + "=" + " " + expr);
308      }
309  }
310

```

표현식이다. 모든 표현식이 여기에 있다. 주석처리된 규칙대로 차례대로 분기문 처리하였다. childCount가 1인 경우는 IDENT와 LITERAL인 경우밖에 없다. 둘다 똑같이 getChild().getText()로 가져오면 되므로 처음 if문에서 동시에 처리해주었다.

그다음 275라인 else if문은 (expr)을 처리하였다.

278라인은 IDENT '[' expr ']'를 처리하였다.

282라인은 IDENT '(' args ')'를 처리하였다.

287라인은 증감연산자와 부호 expr를 처리하였다.

291라인과 292,297라인은 expr 부호(op) expr , IDENT op expr를 처리하였다. isBinaryOperation은 다음과 같다.

```

/*
 * 보조 함수들
 */
//expr + op + expr
boolean isBinaryOperation(MiniParser.ExprContext ctx) {
    return ctx.getChildCount() == 3 && ctx.getChild(1) != ctx.expr();
}

```

303라인은 IDENT '[' expr ']' '=' expr를 처리했다.

15. program

마지막으로 프로그램이다. 프로그램의 시작 부분이다. 프로그램에 들어가기 앞서 decl로 이루어진 program의 값들을 모두 뽑아와서 출력해보았다.

```
12  /*
13   * program : decl+
14   */
15  @Override
16  public void exitProgram(MiniCParser.ProgramContext ctx) {
17
18      String program = "";
19      // 내가 저장한 모든 문자열이 저장
20      for (int i = 0; i < ctx.decl().size(); i++) {
21          program += newTexts.get(ctx.decl(i)) + "\n";
22      }
23      System.out.println(program);
24
25  int max = 500;
26  int arr [2];
27  void main()
28  {
29      int i;
30      int j;
31      int k;
32      int rem;
33      int sum;
34      i = 2;
35      while (i <= max)
36      {
37          sum = 0;
38          k = i / 2;
39          j = i;
40          while (j <= k)
41          {
42              rem = i % j;
43              if(rem == 0)
44              {
45                  sum = sum + j;
46                  ++j;
47              }
48          }
49          if(i == sum)
50          {
51              write(i);
52          }
53          ++i;
54      }
55      int bonus(int a,int b)
56      {
57          return 3;
58      }
59  }
```

모두 잘 출력됨을 볼 수 있으나 들여쓰기가 안되었음을 확인 할 수 있다.
그러므로 program에서 줄바꿈을 할때마다 알맞게 들여쓰기를 해주면된다.
들여쓰기를 구현한 코드는 다음과 같다.

```

9      int dotCount = 0; // 점개수
10     String dot = ""; // 점
11
12     /*
13      * program : decl+
14      */
15     @Override
16     public void exitProgram(MiniCParser.ProgramContext ctx) {
17
18         String program = "";
19         // 내가 저장한 모든 문자열이 저장
20         for (int i = 0; i < ctx.decl().size(); i++) {
21             program += newTexts.get(ctx.decl(i)) + "\n";
22         }
23         StringTokenizer tokenizer = new StringTokenizer(program, "\n");
24         // ....불인 결과값으로 출력하기위해 빈문자열로 재할당
25         program = "";
26         //올바름으로 트oken을 읽어오고 블록을 기준으로 얼마나 들여쓰기를 할지 조절을 해주고 앞에 들여쓰기 개수를 붙여준다.
27         while (tokenizer.hasMoreElements()) {
28             String line = tokenizer.nextToken();
29             if (line.contains("{")) {
30                 program += dot + line + "\n";
31                 setIncreaseDot(); //시작블록이므로 점 4개 추가
32             } else if (line.contains("}")) {
33                 setDecreaseDot(); // 끝 블록이므로 점 4개 감소
34                 program += dot + line + "\n";
35             } else {
36                 program += dot + line + "\n";
37             }
38         }
39         System.out.println(program);
40     }

```



```

388     //점 개수 4개 증가
389     String setIncreaseDot() {
390         dotCount += 4;
391         dot = "";
392         for (int i = 0; i < dotCount; i++) {
393             dot += ".";
394         }
395         return dot;
396     }
397
398     //점 개수 4개 감소
399     String setDecreaseDot() {
400         dotCount -= 4;
401         dot = "";
402         for (int i = 0; i < dotCount; i++) {
403             dot += ".";
404         }
405         return dot;
406     }


```

먼저 점의 개수를 저장할 dotCount와 점을 저장할 String형 dot을 전역변수로 선언해주었다. 그리고 setIncreaseDot()과 setDecreaseDot()이라는 보조함수를 만들었는데 각각 점을 4개 증가 감소해주는 역할을 한다.

program은 decl+로 이루어져있는데 즉 지금까지 데이터를 저장했던 모든 값

들을 여기서 꺼내올 수 있다. 그 꺼내온 값을 String program에 저장한다. 그 후 들여쓰기를 위해 줄바꿈으로 StringTokenizer로 토큰을 나누와 한줄씩 불러온다. 그리고 줄마다 처음에 모두 들여쓰기 개수를 추가해서 다시 program에 저장해줄도록 한다. 시작 블록을 뜻하는 '{'가 포함되어 있다면 dot의 점을 4개 늘리고 '}'가 포함되어 있다면 점을 4개 감소시키는 점(들여쓰기)을 조정해주고 앞에 점 개수만큼 붙여주어 program에 저장해주면 된다. 나머지의 경우는 앞에 점 개수만큼 추가해주면 된다. 그럼 program은 점으로 들여쓰기가 완성될 것이다. 실행결과는 밑과 같다.

[테스트 코드 실행 결과(캡처)]

 test - 메모장

파일(F) 편집(E) 서식(O) 보기(V) 도움말

```
int max = 500; int arr[2];
```

```
void main(){
```

```
int i; int j; int k;
```

```
int rem; int sum;
```

```
    i = 2;
```

```
    while(i <= max) {
```

```
        sum = 0;
```

```
        k = i / 2;
```

```
        j = i;
```

```
        while ( j <= k ) {
```

```
            rem = i % j;
```

```
            if(rem == 0) {
```

```
                sum = sum + j;
```

```
                ++j;
```

```
            }
```

```
            if(i == sum) write(i);
```

```
            ++i;
```

```
        }
```

```
    }
```

```
int bonus(int a, int b){
```

```
    return 3;
```

```
}
```

먼저 test.c 파일은 다음과 같이 예시를 만들었다. 이것을 내가 만든 프로그램으로 돌리면 다음과 같이 결과가 출력된다.

```
int max = 500;
int arr [2];
void main()
{
....int i;
....int j;
....int k;
....int rem;
....int sum;
....i = 2;
....while (i <= max)
....{
.....sum = 0;
.....k = i / 2;
.....j = i;
.....while (j <= k)
.....{
.....rem = i % j;
.....if(rem == 0)
.....{
.....sum = sum + j;
.....++j;
.....}
.....}
.....if(i == sum)
.....{
.....write(i);
.....}
.....++i;
....}
}
int bonus(int a,int b)
{
....return 3;
}
```

잘 출력됨을 볼 수 있다.

[총 과제 수행에 걸린 시간]

3일 걸렸다. 뭐하는 프로그램인지 이해하는데 오래걸리고 완성 후 에러와 예외처리를 하는데도 시간을 꽤 많이 쓴 것 같다.