컴파일러개론 Assign 03

과제 제출일자 : 2019/10/15
학번 : 201502085, 이규봉
분반 : 01
과제 : MiniC Pretty Print

문제 해결 방법

MiniC.g4의 각 문장에 해당하는 부분의 enter, exit 메서드에 해당하는 내용을 작성합니다. 기본적으로 제가 작성한 코드는, enter에서 expr의 내용에 해당하는 노드들에 대해 newTexts에 저장해 놓고 exit 메서드에서 해당 텍스트들을 구문 구조에 따라 배치하는 식으로 동작합니다. 또한, exit 메서드는 MiniC.g4의 각 문장들의 구문 구조의 리프 노드에서 (터미널 노드) 위로 거슬러 올라가며 해당 구문의 newTexts에 저장된 문장을 다시 newTexts에 저장합니다. 최종적으로 exitProgram의 실행 시에 newTexts에 저장된 문장들을 전부 출력하면 test.c의 모든 문장들이 pretty print (원하는 방식으로 구문 구조를 바꿔) 되게 됩니다.

각 exit 메서드의 시작 부분에 *** Description *** 라는 주석과 함께, 아래 exit 메서드에서 사용되는 ctx의 트리가 어떻게 구성되는지를 주석으로 표기해놓았으며, 해당 enter, exit 메서드에서 사용하는 제가 직접 작성한 메서드들에 대해선 모두 *** Description ***라는 주석과 함께, 해당 메서드가 어떤 enter, exit 메서드에서 사용되고 있는지를 나타내었습니다. 코드 내용이 꽤 긴 관계로, 각 메서드 및 트리 구조의 파싱에 대한 상세한 설명은 해당 주석들로 대체하고, 보고서에선 MiniC.g4의 EBNF에 대한 설명과 함께 대략적인 코드의 흐름을 기술하겠습니다.

```
program     : decl+                         ;
```
// 위 program은 해당 문법의 시작 Symbol입니다. decl+이기 때문에 선언이 한 번 이상 반복되어야 하며, 이 중 var_decl은 전역 변수에, fun_decl은 main 함수나 이외의 전역함수들에 해당합니다.

```
decl        : var_decl
            | fun_decl          ;
```
// decl은 전역 변수 및 전역 함수입니다.

```
var_decl    :  type_spec IDENT ';'
            | type_spec IDENT '=' LITERAL ';'
            | type_spec IDENT '[' LITERAL ']' ';'    ;
```
// 전역 변수의 경우, 배열을 선언할 수 있으며, type_spec에서 기술한 타입들은 모두 Literal을 대입 가능함을 알 수 있습니다. 배열 변수엔 = 연산자를 쓸 수 없다는 것도 알 수 있습니다.

```
type_spec   : VOID
            | INT                       ;
```
// 해당 과제의 문법에서 사용하는 타입은 int와 void 뿐입니다.

```
fun_decl    : type_spec IDENT '(' params ')' compound_stmt ;
```
// 전역 함수의 구문에 해당합니다. params는 void이거나 '' (비어있음) 이거나 (param, param..) 과 같은 반복 형태임을 알 수 있습니다. 또한 뒤에 compound_stmt가 이어지므로, 함수 정의에 반드시 '{' '}' 기호가 필요함을 알 수 있

습니다.

```
params          : param (',' param)*
                | VOID
                |                       ;
```

// params는 void이거나 '' (비어있음) 이거나 (param, param..) 과 같은 반복 형태임을 알 수 있습니다.

```
param       : type_spec IDENT
            | type_spec IDENT '[' ']'   ;
```

// 함수의 파라미터에 사용되는 심볼로, 배열 type_spec로 명시한 Literal 대입 가능한 변수들이 이 심볼에 매칭됩니다.

```
stmt        : expr_stmt
            | compound_stmt
            | if_stmt
            | while_stmt
            | return_stmt           ;
```

// stmt는 위 문장들을 모두 포함하는 심볼입니다.

```
expr_stmt   : expr ';'              ;
```

// expr_stmt는 expr뒤에 세미콜론이 붙은 형태입니다. 위 심볼로, expr 뒤에 ;를 붙이는 것만으로 문법에 어긋나지 않는 문장을 만들 수 있다는 것을 알 수 있습니다. 예를 들어, '3 + 3' 은 올바른 문법입니다.

```
while_stmt  : WHILE '(' expr ')' stmt   ;
```

// while문에 해당하는 symbol입니다. 뒤에 중괄호를 굳이 필요로 하지 않기 때문에, while (1) a++; 과 같은 문장도 문법에 어긋나지 않습니다. stmt는 compound_stmt를 포함하기 때문에, 중괄호가 붙어도 올바르게 매칭됩니다.

```
compound_stmt: '{' local_decl* stmt* '}'        ;
```

// 중괄호를 가진 모든 문장들은 위 심볼에 매칭됩니다. local_decl 심볼이 중괄호안에 들어 있으므로, 중괄호 안에서 선언되는 모든 변수들은 local_decl에 해당하며, decl엔 매칭되지 않습니다.

```
local_decl  : type_spec IDENT ';'
            | type_spec IDENT '=' LITERAL ';'
            | type_spec IDENT '[' LITERAL ']' ';'   ;
```

// 지역 변수의 정의입니다.

```
if_stmt     : IF '(' expr ')' stmt
            | IF '(' expr ')' stmt ELSE stmt        ;
```

// if문에 해당하는 심볼입니다. 뒤에 else가 붙어도 올바른 형태입니다. stmt는 compound_stmt를 포함하기 때문에, 중괄호가 붙어도 올바르게 매칭됩니다.

```
return_stmt  : RETURN ';'
             | RETURN expr ';'                    ;
```

// return에 해당하는 심볼입니다. ‘return;, return 1;, return func(1, 2, 3);’ 모두 올바른 문법에 해당합니다.

```
expr    :  LITERAL
        | '(' expr ')'
        | IDENT
        | IDENT '[' expr ']'
        | IDENT '(' args ')'
        | '-' expr
        | '+' expr
        | '--' expr
        | '++' expr
        | expr '*' expr
        | expr '/' expr
        | expr '%' expr
        | expr '+' expr
        | expr '-' expr
        | expr EQ expr
        | expr NE expr
        | expr LE expr
        | expr '<' expr
        | expr GE expr
        | expr '>' expr
        | '!' expr
        | expr AND expr
        | expr OR expr
        | IDENT '=' expr
        | IDENT '[' expr ']' '=' expr            ;
args    : expr (',' expr)*
        |                                        ;
```

// expr은 expr로 재귀적으로 구성되는 케이스를 포함하는 심볼입니다. 따라서, 코드에서는 각 심볼들이 expr을 포함하는 경우, enter 메서드에서 expr에 해당하는 심볼을 newTexts에 put해 놓고 exit에서 가져다 쓰는 식으로 구현하였습니다.

테스트 코드 실행 결과 (캡쳐)

```
"C:\Program Files\Java\jdk1.8.0_221\bin\java.exe" ...
int dd = 2;
int max = 500;
void main(int a, int b, int k[])
{
....int a[30];
....int b = 3;
....int i;
....int j;
....int k;
....int rem;
....int sum;
....i = 2;
....3;
....!2;
....if (1)
........return;
....if (1 != 1)
....{
........return;
....}
....i = ((i + i + i - i * ++i % i) / i);
....a[4] = 5;
....while (!1 != 0xFF)
....{
........sum = 0;
........k = i / 2;
........j = i;
........while (!(j - 3 <= ++k))
........{
............rem = i % j;
............if (rem == 0)
............{
................sum = sum + j;
................++j;
............}
............else
............{
................sum = 1;
............}
........}
........if (i == sum)
............write(i, j);
........++i;
```

```
"C:\Program Files\Java\jdk1.8.0_221\bin\java.exe" ...
int a;
int main()
{
....int x;
....int y;
....if (x > 0)
....{
........x = x + 1;
....}
}
```

pretty print가 각 상황들에 맞게 올바른 출력을 가지고 있는 것을 알 수 있습니다.

총 과제 수행에 걸린 시간

과제를 어떻게 해야 수행할 수 있을지 2 ~ 3 시간 정도 코드를 디버깅 해보며 생각했습니다. 그리고 실제로 구현하는데 3 ~ 4 시간 정도 소요 되었습니다. 그리고 보고서를 작성하는데 30분 ~ 1시간 정도 소요되었습니다.
그 후 질문 게시판에 올라온 if문 뒤에 중괄호가 안 붙는 경우에 대한 답변을 보고 코드를 고치는데, 약간 헤메어 1 ~ 2 시간 정도가 더 소요되었습니다.