컴파일러 Assignment 3

제출 방식:

antlr 파일소스, 화면 캡춰 등을 조교가 지정하는 방식대로 제출하시오.

과제 (개괄):

“antlr를 이용하여 주어진 문법에 대해 C 파일을 pretty print 하는 프로그램을 작성하시오.”

- 블록이나 nesting 되어 들어갈 때는 4칸 들여쓰기
- if 절 등 이나 함수 시작은 다음 줄에 시작 괄호를 표기한다.
- 2진 연산자와 피연산자 사이에는 빈칸을 1칸 둔다.
- 일반 괄호는 expression에 붙여 적는다.

예)

입력

```
if(x>0   ){ x=x   +1;}
```

출력

```
if (x > 0)
{
    x = x + 1;
}
```

주의:

- 문법은 과제 2에서 제시되었던 것을 사용한다.
- 반드시 Listener를 사용한다. (다음 장에 설명)
- 그 외 자세한 사항은 조교가 업로드하는 추가자료를 참고할 것 (문법 등)

참고 사항:

1. 문법 예 (MiniC.g4)

```
grammar MiniC;
program          : decl+              ;
decl             : var_decl
                 | fun_decl           ;
... // 문법만 있고 코드는 없다. (과제2와 달라짐)
```

2. 테스트를 위한 `main` 메소드는 아래와 같다.

```
import org.antlr.v4.runtime.*;
public class TestMiniC {
  public static void main(String[] args) throws Exception
  {
        MiniCLexer lexer = new MiniCLexer( new ANTLRFileStream("test"));
        CommonTokenStream tokens     = new CommonTokenStream( lexer );
        MiniCParser parser           = new MiniCParser( tokens );
        ParseTree tree               = parser.program();

        // 여기부터 새로운 부분
        ParseTreeWalker walker       = new ParseTreeWalker();
        walker.walk(new MiniCPrintListener(), tree );
  }
```

3. ANTLR를 위해 생성된 클래스 MiniCBaseListener를아래와 같이 계승하는 하위클래스 클래스 MiniCPrintListener 를 만든다.

예) 클래스 MiniCBaseListener

```
// Generated from MiniC.g4 by ANTLR 4.4
package grammar;
import org.antlr.v4.runtime.ParserRuleContext;
import org.antlr.v4.runtime.misc.NotNull;
import org.antlr.v4.runtime.tree.ErrorNode;
import org.antlr.v4.runtime.tree.TerminalNode;

/**
 * This class provides an empty implementation of {@link MiniCListener},
 * which can be extended to create a listener which only needs to handle a
subset
 * of the available methods.
 */
public class MiniCBaseListener implements MiniCListener {
        …
        @Override public void
        enterDecl(MiniCParser.DeclContext ctx) {
                /* decl 노드에 preorder로 처리할 부분: 비어있다 */
```

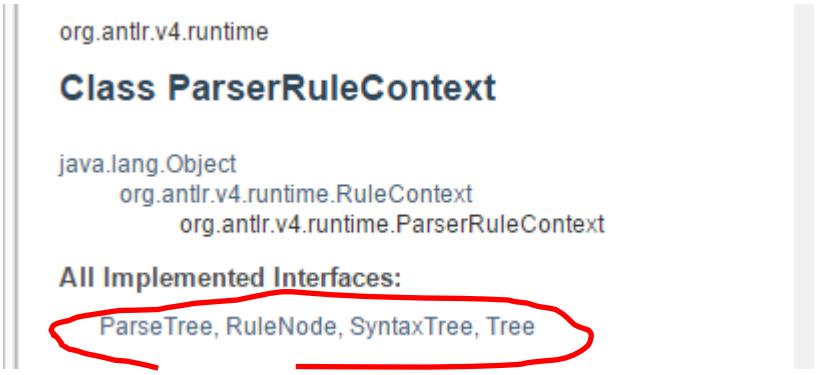
```
    }

    @Override public void
    exitDecl(MiniCParser.DeclContext ctx) {
            /* dcl_specifiers 노드에 preorder로 처리할 부분: 비어있다 */
    }
```

....

- 인터페이스 MiniCListener는 ANTLR가 자동으로 생성한 interface이다. 각 non-terminal에 대해 enter, exit 메소드를 선언하고 있다.
  예) non-terminal인 program에 대해 enterProgram(), exitProgram() 메소드, non-terminal인 var_decl에 대해 enterVar_decl(), exitVar_decl() 메소드 등
- ANTLR의 walker가 walk을 하면 파스트리의 각 노드를 depth-first로 방문하는데, preorder로 수행하고 싶은 것은 enterXXX에, postorder로 수행하고 싶은 것은 exitXXX에 들어있게 된다.
- 클래스 MiniCBaseListener는 ANTLR가 자동으로 생성한 클래스이다. 각 non-terminal에 대해 enterXXX, exitXXX 메소드의 skeleton을 만들어둔다.
- 프로그래머는 MiniCBaseListner를 상속받아 새로운 서브클래스에서 원하는 메소드를 재정의해서 사용한다.

참고 Context

- 자신의 정보 : 위 enter/exit 메소드의 인자인 ctx는 해당 노드의 정보를 모두 가지고 있다. Tree의 노드이기도 하다.



- 자식의 정보: ParseTreeProperty 객체를 하나 써서 ctx와 함께 get()/put()을 사용하게 되면 노드에 내가 원하는 것을 추가로 저장할 수 있다. 이것으로 자식은 부모노드로 전달시키고 싶은 값을 전달한다. (ParseTreeProperty는 Map으로써

키는 tree 노드 (즉, context), 값은 정하고 싶은 타입으로 저장한다.

```
ParseTreeProperty<Integer> values = new ParseTreeProperty<Integer>();
values.put(tree, 36); // tree라는 노드에 36 값 저장
int x = values.get(tree);
values.removeFrom(tree);
```

- 과제에서는 자식노드에서는 put()으로 자신의 ctx에 새로운 text를 저장하고 부모노드에서는 children의 ctx'에 대해 get(ctx')로 읽어올 수 있다.

예)

```
class MiniCPrintListener extends MiniCBaseListener {

    ParseTreeProperty<String> newTexts = new ParseTreeProperty<String>();

    boolean isBinaryOperation(MiniCParser.ExprContext ctx){
            return  ctx.getChildCount() == 3
                        && ctx.getChild(1) != ctx.expr();
                        // '(' expr ')'를 배제
    }
    @Override public void exitExpr( MiniCParser.ExprContext ctx) {
            String s1 = null, s2 = null, op=null;
            if (isBinaryOperation(ctx))
            {
                    // 예: expr '+' expr
                    s1 = newTexts.get(ctx.expr(0));
                    s2 = newTexts.get(ctx.expr(1));
                    op = ctx.getChild(1).getText();
                    newTexts.put(ctx, s1 +" "+ op +" "+ s2);
            }
            else if (...
```

- 파스트리 중 expr → expr '+' expr 로 만들어진 노드가 있다면, ctx.expr(0), ctx.expr(1)이 자식 노드의 ctx들이 된다.
- 부모노드에 값을 전달하는 방법은 recursive decent parser의 리턴 값으로 전달하는 것이 전형적인 방법이었다. 이것은 Visitor 패턴, XML의 DOM 파서와 유사하다. 하지만 지금 Listner 방식은 XML의 SAX 파서와 유사한 방식이다.