# Compiler Project

## Project 2: Parser

정상윤 (2016025032)

## 1. 컴파일 방식 & 환경

Project 1 과 마찬가지로 제공해주신 Makefile 에서 -lfl 부분을 -ll 로 수정하고 그대로 사용하여 터미널에서 make 했습니다. 실습 환경은 macOS Big Sur (v 11.6) 에서 진행하였습니다. 사용된 lex는 2.6.4 버전입니다.

## 2. 구현 방식 설명 & 실행 방법

제공해주신 Hint를 따라가며 구현했습니다.

- **main.c**

NO_PARSE를 FALSE로, NO_ANALYZE를 TRUE로, TraceParse를 TRUE로 수정했습니다.

- **global.h**

우선 yacc 폴더 안 global.h 의 코드의 내용으로 덮어씌워줬습니다. PDF파일 4페이지의 BNF Grammar와 5,6,7페이지의 내용들을 이해하는 것이 제일 중요했습니다

수정된 노드 종류들 (enum)

```
typedef enum {StmtK,ExpK,DclK,PrmK} NodeKind;
typedef enum {VarK,FuncK} DclKind;
typedef enum {IfK,IfElseK,IterK,RetK,CompK} StmtKind;
typedef enum {OpK,ConstK,IdK,AssignK,CallK} ExpKind;
typedef enum {VoidParamK, ParamK} PrmKind;

/* ExpType is used for type checking */
typedef enum {Void,Integer,IntArray,VoidArray} ExpType;
```

수정된 treeNode 구조

```
typedef struct treeNode
   { struct treeNode * child[MAXCHILDREN];
     struct treeNode * sibling;
     int lineno;
     NodeKind nodekind;
     union { StmtKind stmt; ExpKind exp; DclKind dcl; PrmKind prm; } kind;
     union { TokenType op;
             int val;
             char * name;} attr;
     ExpType type; /* for type checking of exps */
   } TreeNode;
```

- **util.c**

globals.h 의 수정된 내용에 맞춰가며 코드 수정이 진행됐습니다. 위 단계에서 StmtKind와 ExpKind 두 개만 사용하는 쪽으로 고쳤기 때문에 다른 함수 추가 없이 printTree 함수만 수정해줘도 됐습니다. global.h 에서 노드들을 정의해주면서 어떻게 파싱해서 출력해줘야할지 대략적으로 파악이 됐기 때문에 큰 어려움은 없었습니다. 처음에 상황에 따라 int, int[], void, void[] 를 출력해주는 구문을 개별적으로 넣어주다가 코드 중복이 심해져서 getTypeString 함수를 추가하여 코드를 간결화 시켰습니다.

- **cminus.y**

우선 yacc 폴더 안 tiny.y 의 코드를 복사한 후 cminus의 symbol들을 definition 섹션에 정의해줬습니다.  BNF Grammar를 잘 이해하여 cminus.y에 이식하는 것이 핵심이었던 것 같습니다. 기존 tiny.y에 있던 코드 내용들과 pdf 파일에 첨부된 예시 코드들을 참고해가며 코드를 작성했습니다. ID를 통해 TreeNode attr.name 에 올바른 값을 넣어주는 처리가 가장 어려웠습니다. 또한 type_specifier로 넘어가 type을 지정해주는데 어려움을 느껴서 type_specifier 문법을 따로 지정해주지 않고 INT/VOID/INT[]/VOID[] 각각 케이스들을 대입했습니다.

### 실행 방법

make clean -> make -> ./cminus_parser {input file}

## 3. 실행 결과

test.1.txt 실행 결과

```
C-MINUS COMPILATION: test.1.txt

Syntax tree:
  Function Declaration: name = gcd, return type = int
    Parameter: name = u, type = int
    Parameter: name = v, type = int
    Compound Statement:
      If-Else Statement:
        Op: ==
          Variable: name = v
          Const: 0
        Return Statement:
          Variable: name = u
        Return Statement:
          Call: function name = gcd
            Variable: name = v
            Op: -
              Variable: name = u
              Op: *
                Op: /
                  Variable: name = u
                  Variable: name = v
                Variable: name = v
  Function Declaration: name = main, return type = void
    Void Parameter
    Compound Statement:
      Variable Declaration: name = x, type = int
      Variable Declaration: name = y, type = int
      Assign:
        Variable: name = x
        Call: function name = input
      Assign:
        Variable: name = y
```

```
        Call: function name = input
      Call: function name = output
        Call: function name = gcd
          Variable: name = x
          Variable: name = y
```

test.2.txt 실행결과

```
C-MINUS COMPILATION: test.2.txt

Syntax tree:
  Function Declaration: name = main, return type = void
    Void Parameter
    Compound Statement:
      Variable Declaration: name = i, type = int
      Variable Declaration: name = x, type = int[]
        Const: 5
      Assign:
        Variable: name = i
        Const: 0
      While Statement:
        Op: <
          Variable: name = i
          Const: 5
        Compound Statement:
          Assign:
            Variable: name = i
              Variable: name = i
            Call: function name = input
          Assign:
            Variable: name = i
            Op: +
              Variable: name = i
              Const: 1
      Assign:
        Variable: name = i
        Const: 0
      While Statement:
        Op: <=
          Variable: name = i
          Const: 4
        Compound Statement:
          If Statement:
            Op: !=
              Variable: name = i
                Variable: name = i
              Const: 0
            Compound Statement:
              Call: function name = output
                Variable: name = i
                  Variable: name = i
```