**Design and Development of Compiler**

**for C- Language**

**(설계 프로젝트 수행 결과)**

1. **Design and Implementation of LALR Parser**

**과목명: [CSE4120] 기초 컴파일러 구성**

**담당교수: 서강대학교 컴퓨터공학과 정 성 원**

**개발자: 정재훈 20151607**

**개발기간: 2019. 5. 1 – 2019. 5. 3**

프로젝트 제목: Design and Development of Compiler for C-Language

Phase 2: Design and Implementation of LALR Parser

제출일 : 2019.05.02

개발자 : 정재훈

**I. 개발 목표**

C- language로 짜여진 코드를 컴파일하는 컴파일러를 만들기 위하여, 두번째 단계인 Parser를 구현하는 프로그램이다. Parser는 Lexical Analysis와 결합하여 Scanner가 Token을 반환할 때마다 Top-Down 방식으로 Parsing을 진행해주고 결과값으로 Abstract syntax tree를 출력해주는 프로그램을 개발한다.

**II. 개발 범위 및 내용**

**가. 개발 범위**

C- Language Parsing을 Top-down 방식으로 구현하기 위해서는 LALR(1) Parser Code를 생성해주는 Bison을 이용해야한다. 따라서 Bison 문법에 맞추어 C- Grammar rule들을 cminus.y에 작성한다. 프로젝트 1에서 구현한 Scanner를 위해 globals.h에 정의한 Token enumtype을 지우고 bison 파일에 token을 표시한다. Syntax tree 구현을 위해 C- 문법을 고려하여 Tree node kind를 세분화하고 각 nodekind에 맞는 tree node를 생성하는 함수를 util.c에 작성한다. 이 분류에 맞추어 cminus.y에 reduce에 따른 Action code를 작성하여 최종 reduce 결과값이 syntax tree의 root node가 나올 수 있도록 한다. Parsing 결과물로서 Syntax tree를 출력하기 위해 util.c의 printTree 함수를 작성한다.

**나. 개발 내용**

**(Parsing을 위한 Rule과 Action 작성)**

Bison이란 LALR(1) context-free grammar로 작성된 Grammar description을 parsing을 위한 C program으로 바꿔주는 툴 프로그램이다. Bison의 포맷은 다음과 같다.

%{

C Declarations

%}

Bison Declarations

%%

Grammar Rules

%%

Additional C Code

이때 Cminus의 문법을 grammar rule 부분에 작성하는데, 이를 기반으로 Shift/ Reduce를 진행한다. Rule마다 Reduce시에 Action을 수행할 수 있어 이를 통해 Syntax tree를 구성할 수 있다. Parsing Stack에 들어있는 각 Terminal이나 Nonterminal은 YYSTYPE이라는 자료형의 data를 가지고 있어 여기에 tree node pointer를 저장하면 Reduce 될 때 right hand의 YYSTYPE data를 가지고 tree를 구성할 수 있다.

**(Tree Node 구성)**

Cminus 문법을 보았을때 각 Tree node는 총 5가지로 구분할 수 있다.

1. Function node : Function Declaration에 사용 parameters, compound statement를 자식으로 가짐

2. Declaration node : Variable Declaration에 사용 Type과 name을 attribute로 가짐

3. Parameter node : Function Declaration에 사용되는 parameter Declaration에 사용 Type과 name을 attribute로 가짐

4. Statement node : Statement는 총 5가지로 분류될 수 있다.

If statement : if-else 조건문에 사용. 조건문과 실행문을 자식으로 가지며 else가 존재하는 경우 else 실행문을 자식으로 가짐.

While statement : While 반복문에 사용. 조건문과 실행문을 자식으로 가짐

Return statement : Return 반환문에 사용. return 값이 존재시 반환하는 expression statement를 자식으로 가짐

Compound Statement : function 서술부에 이용 { } Curly bracket으로 묶여 있으며 여러 local declaration과 statement를 자식으로 가짐

Expression Statement : 수식 표현에 사용. Binary operation과 lefthand side right hand side로 이루어져 있음 Operation은 attribute로 가지고 lefthand와 righthand를 자식으로 가짐

5. Factor node : Expression Statement Leaf node를 담당하며 총 3가지 분류로 나뉠 수 있다.

Id factor : 변수를 의미하는 factor로 name을 가지고 있다.

Arr factor : int array를 의미하는 factor로 name과 index를 자식으로 가지고 있다.

Const factor : const value를 의미하는 factor로 int 상수값을 가지고 있다.

6. Call Node : Function call을 의미하는 tree node로서 call procedure name을 attribute로 가지며 argument는 자식으로 가지고 있다.

**(Print Tree 작성)**

Syntax Tree Node는 해당 node 종류를 알려주는 nodekind , 자식과 형제를 가르키는 child, sibling pointer 그리고 name, value, op같은 attribute가 존재한다. 만약 nodekind가 statement이거나 factor일경우 한단계 더 세부적으로 분류될 수 있으므로 Union type으로 세부 kind 정보를 갖는다.

Print tree는 TreeNode\* 를 인자로 받아, Nodekind를 확인후 각 nodekind 양식에 맞게 해당 노드의정보를 출력한다. 만약 statement나 factor의 경우 세부분류에 따라 다르게 노드정보를 출력해야한다. 그 다음 child를 순회하면서 printTree를 호출하고 sibling으로 node pointer를 이동하여 printTree를 수행한다. 이로서 Left Child Right Sibling을 post order방식으로 순회하며 출력할 수 있다.

**III. 추진 일정 및 개발 방법**

**가. 추진 일정 .**

2019.05.01 ~ 2019.05.02 Parser 코드 작성

2019.05.02 ~ 2019.05.03 보고서 작성

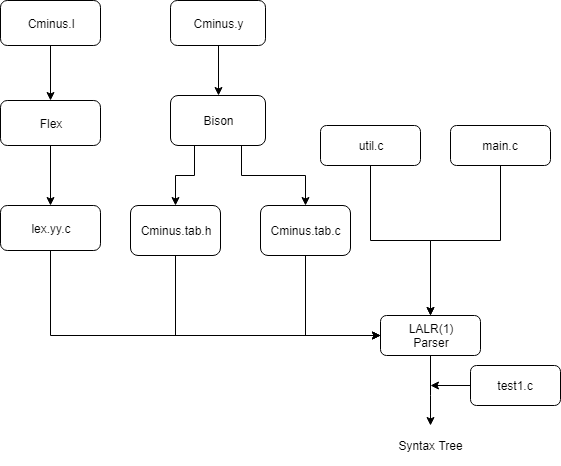
**나. 개발 방법**

project1 : scanner 당시 선언해놓았던 token type enumeration을 globals.h에서 지우고 이를 cminus.y에 따로 token 선언해준다. TreeNode 구조체를 Cminus grammar에 알맞게 고치고 NodeKind 역시 재정의한다. util.c에서는 각 Nodekind에 맞는 TreeNode의 생성함수와 출력 양식에 맞도록 printTree 함수를 작성한다. cminus.y에는 cminus 문법에 맞게 rule을 작성하고, 최종 Accept 단계에서 Syntax tree가 나올 수 있도록 reduce action을 작성한다. 또한 Parsing Error시 에러위치와 토큰을 출력할 수 있도록 에러출력함수를 작성한다. 마지막으로 main함수에 선언되어 있던 NO\_PARSE를 FALSE로 고치고 TraceScan을 FALSE로 바꾸어 token scan출력은 없애고 Syntax tree만 출력 되도록 한다.

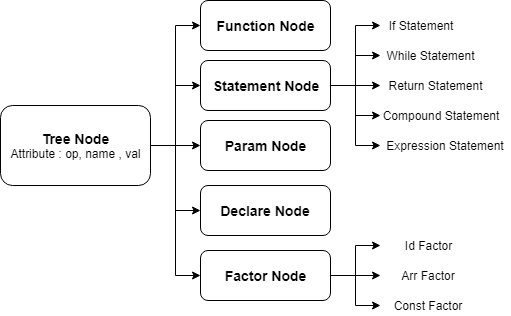
**IV. 연구 결과**

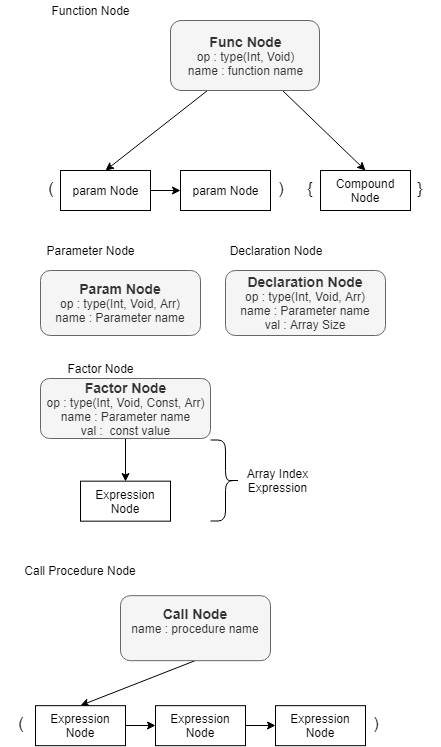
**1. 합성 내용:**

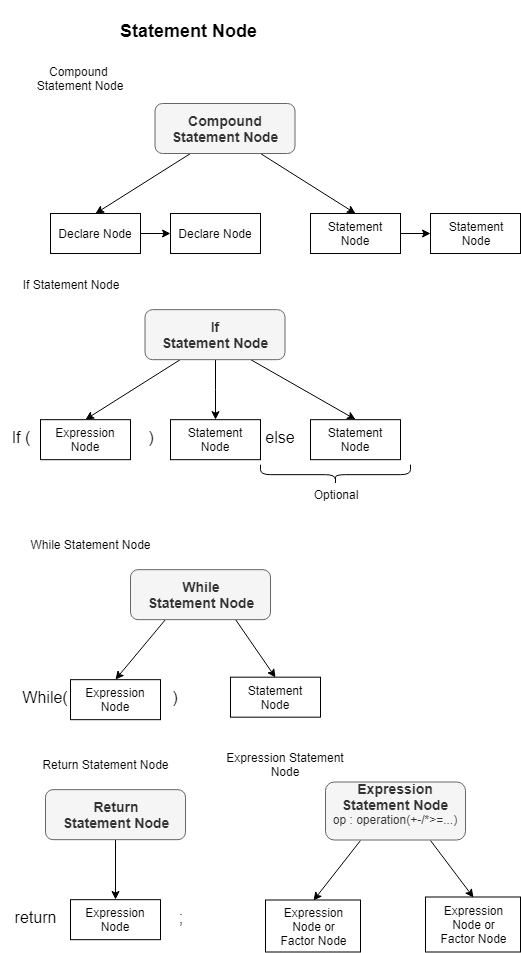
**(프로그램 구성도)**



**(Tree Node 구조)**







**2. 분석 내용**

**(Program Execution)**

Flex를 이용하여 cminus.l을 lex.yy.c파일로 변환하고, Bison을 이용하여 cminus.y를 cminus.tab.h와 cminus.tab.c로 변환한다. lex.yy.c파일에서 yylex()함수가 Scanner 역할을 하며 cminus.tab.c 파일에서 yyparse()함수가 Parser 역할을 한다. yyparse()는 내부적으로 yylex() 를 통해 token을 하나씩 읽어들여 LALR(1) parsing을 수행한다. Parsing이 성공적으로 종료되면 parser는 Syntax tree의 root node pointer를 반환하며 main함수가 이를 받아 util.c에 정의된 printTree함수에 전달한다. printTree함수는 Root node부터 left child right sibling를 순회하며 syntax tree를 출력한다.

**(Operator Precedence and Associativity)**

Bison에서 연산자 우선순위를 정해줄 수 있으나 Cminus grammar는 문법적으로 이를 구현해놓았다. 가장 높은 우선순위를 갖는 연산자가 Tree에서 Leaf node에 가깝게 위치하며 낮은 우선순위를 갖는 연산자가 Tree에서 Root node에 가깝게 위치한다. Cminus grammar에서는 “Syntax Tree 위치상에서 우선순위가 낮은 연산자가 높은 연산자의 위쪽에 위치한다”라는 원리가 지켜지기 때문에 항상 높은 우선순위의 연산자가 먼저 계산되도록 Grammar가 설계되어있다.

우선순위 : relop(<=, <, >, >=, ==, !=) < addop(+, -) < mulop(\*, /)

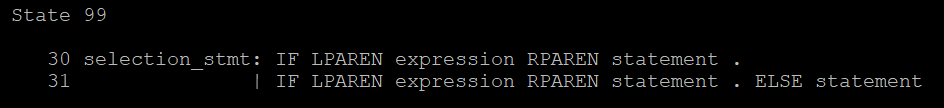
결합법칙의 경우 + - \* / 전부 Left Associativity로 구현해야하는데 grammar rule을 보면 관련 rule들이 전부 left recursion으로 되어있다. 따라서 grammar를 이용하여 Syntax tree가 Left Associativity를 만족할 수 있다.

**(If-Else Dangling Conflict)**

Cminus Grammar에서 If statement에 Shift/Reduce Conflict 가 발생한다.

selection\_stmt -> IF ( expression ) statement | IF ( expression ) statement ELSE statement

in cminus.output



이를 Shift 우선으로 처리하여 Most closely Nested Rule 로서 처리한다. Shift 우선으로 Conflict를 해결하기 위해서는 해당 Rule에 있어서 RPAREN과 ELSE token 사이에 우선순위를 줘야한다.



따라서 cminus.y에 다음과 같이 적어주면 ELSE가 RPAREN보다 우선순위가 높게 되어 다음 State 99 상황에서 next token이 ELSE일 경우 Shift를 하게 되어 Most Closey Nested Rule을 만족하게 된다.

**(Semantic Action)**

Parser에서 Check하는 것은 LALR(1) Grammar인지 보는 것이고 Parser를 통해 생성된 Syntax Tree는 Semantic Analysis에서 이용한다. Semantic Analysis에서 필요한 정보는 변수의 Type, 선언되어 있는 Line number, Operator 종류, 계산된 Value, Data Type 그리고 Identifier의 Name등을 저장해야한다. 이 정보는 Node의 Attribute라고 부르며 Parsing 단계에서 Syntax Tree를 생성할 때 Semantic Analysis를 위해 Attribute도 같이 저장해주어야 한다. 이를 Semantic Action이라고 한다.

Bison에서는 이를 위해 Reduce Action Part를 작성할 수 있다. Syntax Tree의 Pointing 구조는 righthand Nonterminal의 return value로 쉽게 생성이 가능하지만, Name이나 Line number, Value는 조금 다른 방법이 필요하다.

Name은 ID token Reduce시에 recognize 되는데 reduce 되기 전까지 이를 저장하고 있어야 Attribute로서 저장할 수 있다. 최종적으로 한 rule이 reduce되기까지 여러개의 ID가 등장할 수 있으므로 이를 저장하기 위해서는 Stack 자료구조가 필요하다. 따라서 이를 해결하기 위하여 identifier -> ID라는 룰을 하나 추가하고 reduce action으로서 Stack에 ID와 해당 line number을 push해 놓는다. 만약 grammar rule이 reduce될때 id 값을 attribute로 저장해야 하는 상황에는 stack pop하여 나온 id와 line number을 attribute로 갖는다.

Number의 경우 역시 마찬가지로 Stack을 이용해야 하지만 Cminus grammar에서 NUM token이 포함된 rule에서는 NUM 이 여러번 등장하지 않는다. 따라서 Stack을 사용하지 않고 NUM token 오른쪽에 바로 reduce action으로 임시변수 SavedNum에 number를 저장하여 NUM이 포함된 rule이 reduce 될 때 number을 해당 노드 val attribute에 저장시킨다.

Operator는 Calulate Operator 과 Boolean Operator 두가지가 있는데 Calculate Operator는 addop mulop 이 두 NonTermnial로 reduce된다. Boolean Operator는 relop로 reduce된다. Operator는 따로 Node를 생성할 이유가 없고 Expression Node에 op attribute로 저장해야한다. 따라서 addop mulop relop의 reduce action으로 $$에 TreeNode\* 를 전달해야하지만 Typecasting하여 해당 Operator Token enum value를 전달하도록 한다.

Type Specifier(INT VOID)도 operator와 마찬가지로 Type Specifier Token enum value를 전달하도록 한다.

**3. 제작 내용**

**(Tree Node 정의 및 enum type 삭제)**

globals.h에 Cminus grammar을 위한 Tree Node Kind를 합성 내용처럼 분류하고 정의해놓았다. 또한 Project 1에서 정의해놓았던 Token의 Enumtype을 제거하고 선언부만 남겨둔다.

**(LALR Parser를 위한 bison 파일 제작)**

cminus.y에 Cminus Grammar을 정의하고 그에 따른 Semantic Action을 작성한다. 이때 ID, Line Number Semantic Action을 위한 Stack을 생성하였다. 삭제했던 Token을 이 파일에 재정의하고 IF-ELSE Conflict 제거를 위한 Precedence와 Syntax Error 출력함수를 정의한다.

**(Syntax Tree 출력을 위한 PrintTree 작성)**

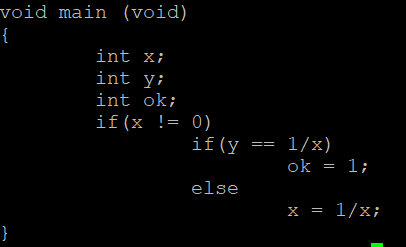
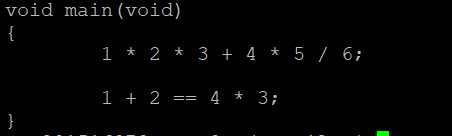
util.c에 Left Child Right Sibling을 순회할 수 있도록 printTree()함수를 작성하였다. 각 node마다 적절한 semantic attribute를 출력하고 Indent와 Unindent를 적절히 활용하여 Tree 구조를 보다 명확하게 볼 수 있도록 하였다. 보다 토큰 출력을 간편하게 하기위해 Token Name은 static char\* 배열에 저장하여 enum value 순서에 따라 저장하였다.

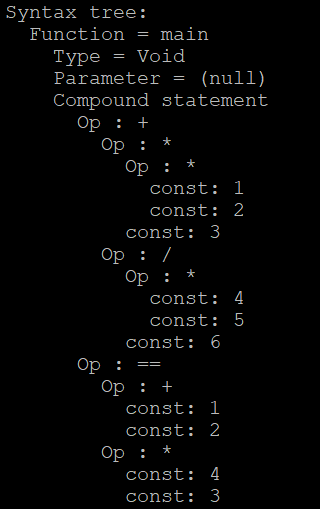
**(Parsing 진행과 Syntax Tree 출력)**

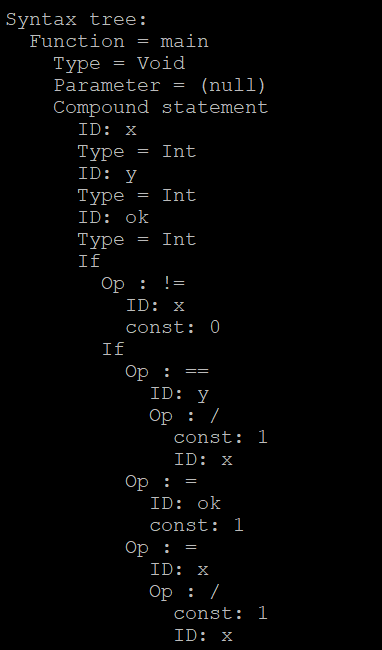
Project1에서 main.c에선 Token Scanner을 위한 Lexical Analysis 진행상황만을 출력했는데 NO\_PARSE를 FALSE로 바꾸어 Parsing이 진행되도록 한다. TraceScan을 FALSE로 바꾸어 Lexical Analysis 진행상황은 출력되지않게 하고 TraceParse는 TRUE로 두어 printTree()함수를 호출할 수 있도록 한다.

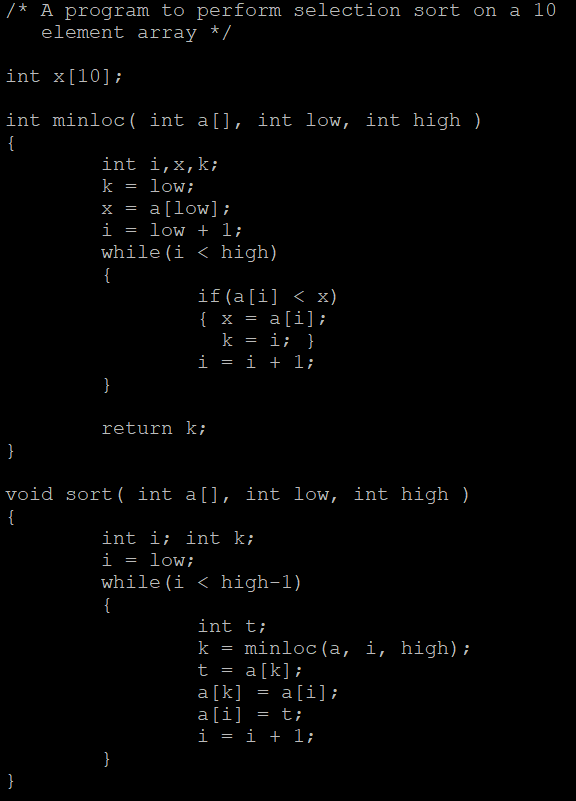
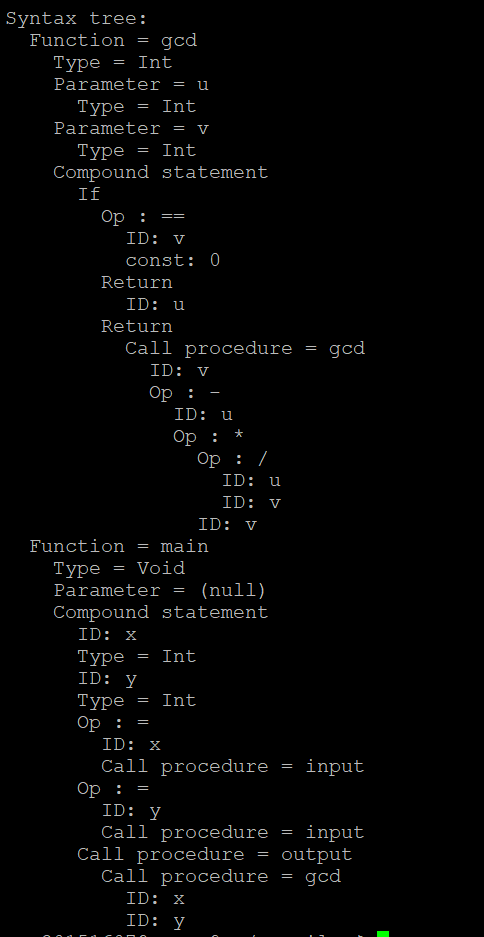
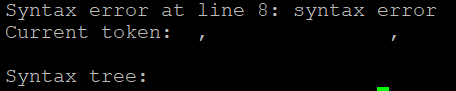
**4. 시험 내용**

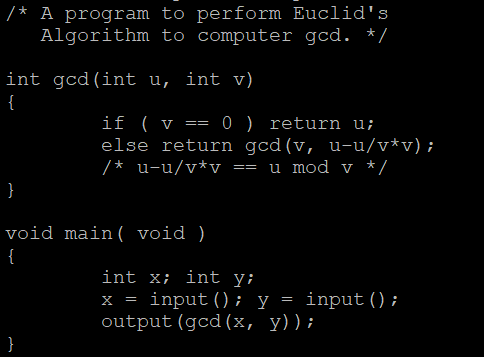
**(Dangling Else Problem) (Precedence and Associativity)**







**(None Error gcd algorithm) (Error Code)**



**5. 평가 내용**

Cminus grammar의 기존 있었던 Shift/Reduce를 잘해결하여 Conflict를 해소하였다. 다음 Project인 Semantic Analysis을 위해 필요한 Tree Node 분류와 Attribute 분류를 미약하게나마 수행하였으며 ID, Lineno, Type등은 완성도 높게 구현하였다. 하지만 현 Project는 Syntax Tree Print가 목적이기에 Value, Type등의 Attribute 계산 및 전달은 남겨두었다. Indent를 이용하여 Syntax Tree를 보다 쉽게 볼 수 있도록 하였고, 각 Tree Node에 따른 정보 출력 및 Syntax Error 출력도 문제없이 작동하였다.

**V. 기타**

**1. 자체 평가**

Grammar Rule 작성할때, ID와 Line number을 Stack에 보다 간편하게 넣기 위해 Macro 함수를 이용하였고, ID는 따로 새로운 룰 Identifier -> ID 를 추가시켜, 가독성과 코드량을 획기적으로 줄일 수 있었다. Print Tree에서 Token Name 출력이 필요한데 이는 Token Name들을 util.c에 static char \* 배열로 저장하여 Enumeration value만 알면 해당 Token 이름을 쉽게 출력할 수 있다. 여기서 NONE 토큰을 하나 더 추가했는데 이는 Bison에 의해 Token Enum value가 생성될 때, 맨 처음 값을 NONE으로 지정하여 Token Name 배열에 접근을 용이하도록 하기위해서이다.

**2 . 느낀점**

이번 프로젝트를 하면서 수업시간에 이론적으로 다루었던 LALR(1) Parsing을 실제로 구현해보는 경험을 할 수 있었다. 물론 Bison 이라는 Tool을 이용해 해당 Parser을 만들었지만, Biosn Option인 YYDEBUG를 통해 실제 코드가 Parsing되는 과정을 눈으로 볼 수 있었고, -v 옵션을 통해 State의 Item들과 Reduce Shift Handle을 보고 Conflict를 해결하는 방법을 모색해볼 수 있었다. 따라서 이번 project 2를 통해 이론적 지식을 실제로 적용해보면서 Compiler를 좀 더 구체화시키는 경험이 된 것 같다.