2. Parser

Compilation method & Environment

Compilation Method

```
CC = gcc
CFLAGS =

OBJS = main.o util.o scan.o parse.o symtab.o analyze.o code.o cgen.o
OBJS_SCANNER = main.o util.o scan.o globals.h
OBJS_FLEX = main.o util.o lex.yy.o globals.h
OBJS_BISON = main.o util.o lex.yy.o symtab.o y.tab.o globals.h

cminus: $(OBJS_SCANNER)
    $(CC) $(CFLAGS) $(OBJS_SCANNER) -o cminus

cminus_flex: $(OBJS_FLEX)
    $(CC) $(CFLAGS) main.o util.o lex.yy.o -o cminus_flex -lfl

lex.yy.o: cminus.l scan.h util.h globals.h
    flex cminus.l
    $(CC) $(CFLAGS) -c lex.yy.c -lfl

cminus_bison: $(OBJS_BISON)
    $(CC) $(CFLAGS) $(OBJS_BISON) -o cminus_bison -lfl

y.tab.o: cminus.y globals.h
    bison -d cminus.y --yacc
    $(CC) $(CFLAGS) -c y.tab.c

clean_cminus:
    rm cminus main.o util.o scan.o

clean_cminus_flex:
    rm cminus_flex main.o util.o lex.yy.o

clean_cminus_bison:
    rm cminus_bison main.o util.o lex.yy.o symtab.o y.tab.o
```

```
youngseo@YS-VirtualBox:~/Desktop/2020_ELE4029_2016024784/2_Parser$ make clean_cminus_bison
rm cminus_bison main.o util.o lex.yy.o symtab.o y.tab.o
youngseo@YS-VirtualBox:~/Desktop/2020_ELE4029_2016024784/2_Parser$ make y.tab.o
bison -d cminus.y --yacc
cminus.y: warning: 1 shift/reduce conflict [-Wconflicts-sr]
gcc  -c y.tab.c
y.tab.c: In function ‘yyparse’:
y.tab.c:2065:7: warning: implicit declaration of function ‘yyerror’; did you mean ‘yyerrok’
? [-Wimplicit-function-declaration]
 2065 |       yyerror (YY_("syntax error"));
      |       ^~~~~~~
      |       yyerrok
youngseo@YS-VirtualBox:~/Desktop/2020_ELE4029_2016024784/2_Parser$ make cminus_bison
gcc    -c -o main.o main.c
gcc    -c -o util.o util.c
flex cminus.l
gcc  -c lex.yy.c -lfl
gcc    -c -o symtab.o symtab.c
gcc  main.o util.o lex.yy.o symtab.o y.tab.o globals.h -o cminus_bison -lfl
```

1. make clean_cminus_bison

   위 명령어를 통한 기존 파일 삭제

2. make y.tab.o

   위 명령어를 통해 y.tab.h 생성

3. make cminus_bison

   위 명령어를 통해 cminus_bison 이름의 실행 파일 생성

Environment

Ubuntu 20.04.1 LTS

Implementation & How to operate

main.c, globals.h, util.c, util.h cminus.y를 수정

1) main.c

```
10    /* set NO_PARSE to TRUE to get a scanner-only compiler */
11    #define NO_PARSE FALSE
12    /* set NO_ANALYZE to TRUE to get a parser-only compiler */
13    #define NO_ANALYZE TRUE
```

```
39    /* allocate and set tracing flags */
40    int EchoSource = FALSE;
41    int TraceScan = FALSE;
42    int TraceParse = TRUE;
43    int TraceAnalyze = FALSE;
44    int TraceCode = FALSE;
45
46    int Error = FALSE;
```

Parser 실행을 위해 위의 사진과 같이 main.c 수정

2) globals.h

우선 기존 globals.h의 내용을 yacc 폴더의 globals.h로 변경

```
typedef enum {StmtK,ExpK,DeclK,ParamK,TypeK} NodeKind;
typedef enum {CompK,IterK,SelectK,RetK} StmtKind;
typedef enum {OpK,ConstK,IdK,AssignK,ArrIdK,CallK} ExpKind;
typedef enum {VarK,ArrVarK,FuncK} DeclKind;
typedef enum {SingleParamK,ArrParamK} ParamKind;
typedef enum {TypeNameK} TypeKind;

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

Syntax Tree parsing을 위해, BNF에 필요한 Decl, Param, Type을 나타내는 노드들을 **NodeKind**에 추가

각 Node의 kind는 BNF를 참고하여 C-MINUS를 위해 필요한 요소들로 재구성

예를 들면, DeclKind에는 변수 선언(VarK), 배열 변수 선언(ArrVarK), 함수 선언(FuncK) 등이 있습니다

```
77   typedef struct ArrayAttr {
78       TokenType type;
79       char * name;
80       int size;
81   } ArrayAttr;
82
83   typedef struct treeNode {
84       struct treeNode * child[MAXCHILDREN];
85       struct treeNode * sibling;
86       int lineno;
87       NodeKind nodekind;
88
89       union {
90           StmtKind stmt;
91           ExpKind exp;
92           DeclKind decl;
93           ParamKind param;
94           TypeKind type;
95       } kind;
96
97       union {
98           TokenType op;
99           TokenType type;
100          int val;
101          char * name;
102          ArrayAttr arr;
103      } attr;
104
105      ExpType type; /* for type checking of exps */
106  } TreeNode;
```

위에서 변경 된 문법적 요소들을 저장할 수 있도록, TreeNode 구조체 수정

ArrayAttr은 배열 변수를 저장하기 위한 새로운 구조체 입니다

3) util.c & util.h

```
29   TreeNode * newDeclNode(DeclKind);
30
31   TreeNode * newParamNode(ParamKind);
32
33   TreeNode * newTypeNode(TypeKind);
```

util.h에 새롭게 추가된 Node 생성을 위한 함수 선언

```
101   TreeNode * newDeclNode(DeclKind kind) {
102     TreeNode * t = (TreeNode *) malloc(sizeof(TreeNode));
103     int i;
104     if (t==NULL)
105       fprintf(listing,"Out of memory error at line %d\n",lineno);
106     else {
107       for (i=0;i<MAXCHILDREN;i++) t->child[i] = NULL;
108       t->sibling = NULL;
109       t->nodekind = DeclK;
110       t->kind.decl = kind;
111       t->lineno = lineno;
112     }
113     return t;
114   }
```

기존에 정의되어 있는 함수를 이용하여, util.h에 정의한 함수를 구현합니다

함수 간의 차이점은 nodekind를 함수에 들어오는 node의 type에 맞게 설정하며,

kind를 매개 변수로 받아 해당 node 중 어떤 종류 인지를 설정합니다

위의 사진은 추가된 node 중 Declaration Node에 대한 함수입니다 (newParamNode, newTypeNode 함수 역시 동일한 방법으로 동작)

```
191       if (tree->nodekind == StmtK)
192       { switch (tree->kind.stmt) {
193           case CompK:
194             fprintf(listing, "Compound statement :\n");
195             break;
196           case IterK:
197             fprintf(listing, "While (condition) (body)\n");
198             break;
199           case SelectK:
200             if(tree->child[2] == NULL)
201               fprintf(listing, "If (condition) (body)\n");
202             else
203               fprintf(listing, "If (condition) (body) (else)\n");
204             break;
205           case RetK:
206             fprintf(listing, "Return: \n");
207             break;
208           default:
209             fprintf(listing, "Unknown ExpNode kind\n");
210             break;
211         }
212       }
```

printTree() 함수를 수정하여 C-MINUS 문법에 알맞은 Syntax Tree 출력으로 변경 및 추가 작업 수행

printTree() 함수는 nodekind와 해당 노드 내부에서의 역할에 따라, 그에 맞는 출력을 할 수 있게 구현

위의 사진은, 그 중 일부인 Statement Node에 대한 수정

```
188         if (tree->nodekind != TypeK)
189           printSpaces();
```

또한 printTree() 함수 수정 중 Type Node를 출력하는 경우, INDENT가 없어야 하기 때문에 초반에 조건문 추가

4) cminus.y

tiny.y의 내용을 GNF에 기반하여 전면 수정하는 작업을 수행

1 - Definition

```
24   /* Keywords */
25   %token IF ELSE WHILE RETURN INT VOID
26   /* ID & NUM */
27   %token ID NUM
28   /* Special Symbols */
29   %token PLUS MINUS TIMES OVER GT GE LT LE EQ NE ASSIGN LPAREN RPAREN LBRACE RBRACE LCURLY RCURLY SEMI COMMA
30   %token ERROR
```

정의부에 Token들을 새롭게 정의

2 - Rules

BNF syntax에 필요한 Rule들을 정의하였습니다. 많은 rule 중 몇가지 간단한 예시와 설명이 필요한 부분에 대해 언급하겠습니다

```
34   program             : declaration_list { savedTree = $1;}
35                       ;
36
37   declaration_list    : declaration_list declaration
38                         {
39                           if ($1 != NULL)
40                           { YYSTYPE node = $1;
41                             while (node->sibling != NULL)
42                               node = node->sibling;
43                             node->sibling = $2;
44                             $$ = $1;
45                           }
46                           else $$ = $2;
47                         }
48                       | declaration { $$ = $1; }
49                       ;
50
51   declaration         : var_declaration { $$ = $1; }
52                       | fun_declaration { $$ = $1; }
53                       ;
54
55   identifier          : ID { savedName = copyString(tokenString); }
56                       ;
```

program을 보면, 현재 $1인 declaration_list가 savedTree에 할당됩니다

declaration을 보면, $$는 규칙의 LHS를 나타내기 때문에 declaration 입니다. 즉 여기서는 declaration에 상황에 맞게 var_declaration, fun_declaration 중 하나를 할당하라는 의미 입니다

```
55   identifier          : ID { savedName = copyString(tokenString); }
56                       ;
57
58   number              : NUM { savedNumber = atoi(tokenString); }
59                       ;
```

PDF의 BNF에는 없는 규칙을 정의하여 추가하였습니다

위의 규칙 추가 없이는 tokenString의 값이 계속 갱신 되기 때문에, ID와 NUM의 값이 유실됩니다. 따라서 token 처리 전에 savedName과 savedMumber에 원하는 값을 미리 저장해 둘 수 있도록 하였습니다

```
61   var_declaration     : type_specifier identifier SEMI
62                         {
63                           $$ = newDeclNode(VarK);
64                           $$->attr.name = savedName;
65                           $$->child[0] = $1;
66                         }
67                       | type_specifier identifier LBRACE number RBRACE SEMI
68                         {
69                           $$ = newDeclNode(ArrVarK);
70                           $$->attr.arr.name = savedName;
71                           $$->attr.arr.size = savedNumber;
72                           $$->child[0] = $1;
73                         }
74                       ;
```

var_declaration(= $$)에 적절한 Node를 생성하여 준 뒤, 정보를 입력합니다. 마지막으로 child에 변수 타입을 저장합니다

```
146    compound_stmt       : LCURLY local_declaration statement_list RCURLY
147                          {
148                            $$ = newStmtNode(CompK);
149                            $$->child[0] = $2;
150                            $$->child[1] = $3;
151                          }
152                        ;
```

compound_stmt(= $$)에 CompK 타입의 노드를 만들어 할당 후,
local_declaration과 statement_list를 자식 노드에 추가하라는 의미의 코드입니다

Example and Result Screenshot

test.1.cm의 내용 및 결과

```
/* A program to perform Euclid's
   Algorithm to computer gcd */

int gcd (int u, int v)
{
        if (v == 0) return u;
        else return gcd(v,u-u/v*v);
        /* u-u/v*v == u mod v */
}

void main(void)
{
        int x; int y;
        x = input(); y = input();
        output(gcd(x,y));
}
```

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

Syntax tree:
  Function declaration, name : gcd, return type : int
    Single parameter, name : u, type : int
    Single parameter, name : v, type : int
    Compound statement :
      If (condition) (body) (else)
        Op : ==
          Id : v
          Const : 0
        Return:
          Id : u
        Return:
          Call, name : gcd, with arguments below
            Id : v
            Op : -
              Id : u
              Op : *
                Op : /
                  Id : u
                  Id : v
                Id : v
  Function declaration, name : main, return type : void
    Single parameter, name : (null), type : void
    Compound statement :
      Var declaration, name : x, type : int
      Var declaration, name : y, type : int
      Assign : (destination) (source)
        Id : x
        Call, name : input, with arguments below
      Assign : (destination) (source)
        Id : y
        Call, name : input, with arguments below
      Call, name : output, with arguments below
        Call, name : gcd, with arguments below
          Id : x
          Id : y
```

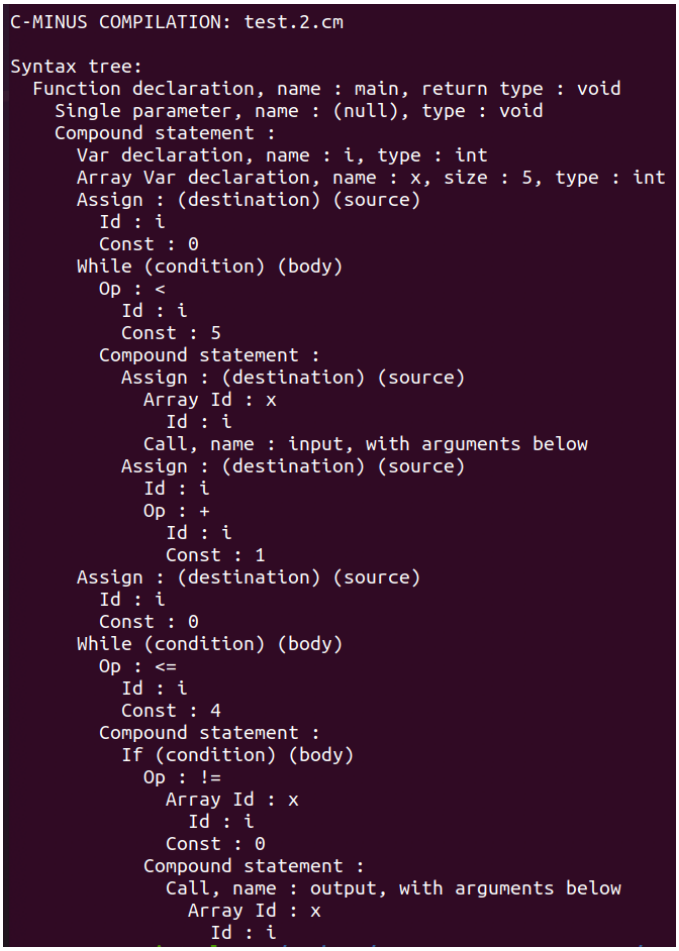
Test.2.cm의 내용 및 결과

```
void main(void)
{
        int i; int x[5];

        i = 0;
        while( i < 5 )
        {
                x[i] = input();

                i = i + 1;
        }

        i = 0;
        while( i <= 4 )
        {
                if( x[i] != 0 )
                {
                        output(x[i]);
                }
        }
}
```



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

Syntax tree:
  Function declaration, name : main, return type : void
    Single parameter, name : (null), type : void
    Compound statement :
      Var declaration, name : i, type : int
      Array Var declaration, name : x, size : 5, type : int
      Assign : (destination) (source)
        Id : i
        Const : 0
      While (condition) (body)
        Op : <
          Id : i
          Const : 5
        Compound statement :
          Assign : (destination) (source)
            Array Id : x
              Id : i
            Call, name : input, with arguments below
          Assign : (destination) (source)
            Id : i
            Op : +
              Id : i
              Const : 1
      Assign : (destination) (source)
        Id : i
        Const : 0
      While (condition) (body)
        Op : <=
          Id : i
          Const : 4
        Compound statement :
          If (condition) (body)
            Op : !=
              Array Id : x
                Id : i
              Const : 0
            Compound statement :
              Call, name : output, with arguments below
                Array Id : x
                  Id : i
```