Compiler Project02_Parser Report

2015004693_양상헌

Goal:

- **Implementation of C-MINUS Parser by using 'Yacc'**

Process:

- **By exchanging default "globals.h" to "globals.h" in 'yacc' directory, and editing it.**
- **By editing "yacc/tiny.y" to make "cminus.y". (by 'Yacc', "cminus.y" becomes "y.tab.c" which is an alternative of "parser.c" in default.**
- **By editing "main.c" and "util.c" in order to print appropriate Parsing Tree.**
- **Editing Makefile to generate "cminus" executable file that works as a C-Minus Parser.**

(1) Overwrite "loucomp/yacc/globals.h" to default "loucomp/globals.h" and edit it.

가장먼저 과제 명세에 나와있는 설명대로 기존의 "globals.h"를 'yacc' 폴더안에 있는 "globals.h"로 바꿔준 후 아래와 같이 편집하였다.

```
/*Edited globals.h (변경된 부분은 빨간색으로 표시)*/
. . .
/* MAXRESERVED = the number of reserved words */
#define MAXRESERVED 12
. . .
/************  Syntax tree for parsing ************/
typedef enum {StmtK,ExpK} NodeKind;
typedef enum {IfK, RepeatK,AssignK,ReadK,WriteK, CompndK,SelectK, IterK, ReturnK, CallK }
StmtKind;
typedef enum {VarK, ArrVarK, FuncK, ParamK, ArrParamK, RelOpK, OpK,ConstK,IdK} ExpKind;
/* ExpType is used for type checking */
typedef enum {Void,Integer,Boolean} ExpType;
#define MAXCHILDREN 3
typedef struct treeNode
   { struct treeNode * child[MAXCHILDREN];
     struct treeNode * sibling;
     int lineno;
     NodeKind nodekind;
     union { StmtKind stmt; ExpKind exp;} kind;
     union { TokenType op;
             int val;
             char * name;} attr;
     ExpType type; /* for type checking of exps */
        int isParam; //1 or 0;
        int isArr; //0 or Sizeof array.
   } TreeNode;
```

일단 MAXRESERVED 를 기존의 8에서 12로 변경해주었다. 사실 이 숫자를 6 혹은 8 로 해줘도 큰 에러 없이 잘 돌아가긴 하지만, Project01 의 환경을 최대한 보존한 상태에서 Project02를 수행하는 것이 좋다고 판단하여 12로 해주었다. (실제로 CMINUS 에서 사용하는 reserved word는 총 6개이다.)

또한 Parsing tree의 Node를 알맞게 생성, 분류하기 위해 StmtKind 와 ExpKind 에 새로운 타입을 추가해 주었다. 이를 통해 추후에 ParsingTree를 Print 할 때 올바른 출력 결과를 얻을 수 있다. Parsing Tree를 Print 해주는 부분은 "util.c"편집 부분에서 자세히 다루도록 하겠다.

또한 treeNode 구조체에 isParam 과 isArr 변수를 추가하여, 배열 변수가 선언되었을 때, 배열의 크기 정보를 저장하고, 현재 Node가 함수의 parameter인지 아닌지에 관한 정보를 저장하게끔 하였다.

(2) Copy “loucomp/yacc/tiny.y” to “loucomp/” and rename it “cminus.y” and edit it.

-cminus.y 에서 사용한 Context-Free Grammar.

```
Program           ->      declaration_list
declaration_list  ->      declaration_list declaration | declaration
declaration       ->      var_declaration | fun_declaration
var_declaration   ->      INT saveid SEMI | INT saveid LBRACE savenum RBRACE SEMI
                          | VOID saveid SEMI | VOID saveid LBRACE savenum RBRACE SEMI
saveid            ->      ID
savenum           ->      NUM
fun_declaration   ->      INT saveid LPAREN params RPAREN compound_stmt
                          | VOID saveid LPAREN params RPAREN compound_stmt
params            ->      param_list
param_list        ->      param_list COMMA param | param
param             ->      INT saveid | INT saveid LBRACE RBRACE
                          | VOID saveid | VOID saveid LBRACE RBRACE
compound_stmt        ->   LCURLY local_declarations statement_list RCURLY
local_declarations   ->   local_declarations var_declaration | epsilon
statement_list       ->   statement_list statement | epsilon
statement            ->   expression_stmt | compound_stmt | selection_stmt
                          | iteration_stmt | return_stmt
expression_stmt    ->     expression SEMI | SEMI
selection_stmt     ->     IF LPAREN expression RPAREN statement %prec LOWER_THAN_ELSE
                          | IF LPAREN expression RPAREN statement ELSE statement
iteration_stmt     ->     WHILE LPAREN expression RPAREN statement
return_stmt        ->     RETURN SEMI | RETURN expression SEMI
expression         ->     var ASSIGN expression | simple_expression
var                ->     saveid | saveid LBRACE expression RBRACE
simple_expression  ->     additive_expression LE additive_expression
                          | additive_expression LT additive_expression
                          | additive_expression GT additive_expression
                          | additive_expression GE additive_expression
                          | additive_expression EQ additive_expression
                          | additive_expression NE additive_expression
                          | additive_expression
additive_expression   ->  additive_expression PLUS term | additive_expression MINUS term | term
term              ->      term TIMES factor | term OVER factor | factor
factor            ->      LPAREN expression RPAREN | var | call | NUM
call              ->      saveid LPAREN args RPAREN
args              ->      arg_list | epsilon
arg_list          ->      arg_list COMMA expression | expression
```

위의 CFG를 바탕으로 “cminus.y”를 편집하였다. 처음엔 과제 명세에 나와있는 CFG와 똑같은 CFG로 구현하였지만, 여러부분에서 오류가 발생하여 오류가 발생하지 않도록 CFG를 조금 고쳤고, 위의 CFG가 최종적으로 사용한 CFG이다.

- 과제 명세에 나와있는 CFG와 크게 다른 점은 다음과 같다.

1. ID와 NUM에서 문자열과 숫자를 저장하기 위해 saveid, savenum이라는 non-terminal을 새롭게 만들었다. 처음엔 그냥 ID , NUM을 바로 사용하였는데 그렇게 되면 reduce 과정에서 ID에 필요한 변수이름이나 NUM에 필요한 숫자를 읽지못하는 경우가 발생하여 이렇게 고쳐서 진행하였다.
2. type_specifier 라는 non-terminal을 없애고 production을 INT 와 VOID 두가지로 나누어서 진행하였는데 이는 type 정보를 child 노드로 갖지 않고 현재 노드에 갖게 하기 위함이다. 처음엔 type_specifier를 child 노드에 저장을 하였는데 그렇게 되면 Parsing Tree를 Print하는 과정에서 type 이 다음 줄에 출력되는 일이 발생하여 이를 해결하고자 이렇게 고쳐서 진행하였다.
3. relop, op 와 같은 연산자를 나타내는 non-terminal들도 없애고 각각 LE, LT, GT, GE, EQ, NE 와 PLUS, MINUS, TIMES, OVER 로 non-terminal을 거치지 않고 terminal을 사용해서 production을 진행 하였는데, 이는 reduce작업이 일어날 때 operand들을 효과적으로 child노드에 바로 저장하기 위함이다. 이렇게 하지 않으면 reduce과정에서 operand들을 child노드로 만드는 것이 매우 힘들었기 때문에 이렇게 고쳐서 진행하였다.
4. Dangling Else를 해결하기 위해 “%prec LOWER_THAN_ELSE”를 production에 추가하여 주었다. 이것을 추가하기 전엔 빌드 과정에서 selection_stmt 에서 shift/reduce conflict가 발생한다. 이 conflict를 해결하기 위하여 non-terminal에 %prec을 이용하여 Precidence를 부여하였다.

-<편집된 cminus.y (1),(2)>

(1)

```
. . .
%{
. . .
static char * savedName; /* for use in assignments */
static int savedNumber;
static int savedLineNo;  /* ditto */
. . .
%}
%token IF ELSE WHILE RETURN INT VOID THEN END REPEAT UNTIL READ
WRITE
%token ID NUM
%token ASSIGN EQ NE LT LE GT GE PLUS MINUS TIMES OVER
%token LPAREN RPAREN LBRACE RBRACE LCURLY RCURLY SEMI COMMA
%token ERROR
%nonassoc LOWER_THAN_ELSE
%nonassoc ELSE

%% /* Grammar for CMINUS */
program     : declaration_list { savedTree = $1;}
            ;
declaration_list : declaration_list declaration {
                        YYSTYPE t = $1;
                        if(t != NULL){
                          while(t->sibling != NULL){
                            t = t->sibling;
                          }
                          t->sibling = $2;
                          $$ = $1;
                        }
                        else { $$ = $2;}
                        }
            | declaration { $$ = $1; }
            ;
declaration : var_declaration { $$ = $1; }
            | fun_declaration { $$ = $1; }
            ;
var_declaration: INT saveid SEMI {/*single int var*/
                        $$ = newExpNode(VarK);
                        $$->lineno = lineno;
                        $$->attr.name = savedName;
                        $$->type = Integer;
                        }
            | INT saveid LBRACE savenum RBRACE SEMI {
                        /*int array var*/
                        $$ = newExpNode(ArrVarK);
                        $$->lineno = lineno;
                        $$->attr.name = savedName;
                        $$->isArr = savedNumber;
                        $$->type = Integer;
                        }
            | VOID saveid SEMI {/*single void var*/
                        $$ = newExpNode(VarK);
                        $$->lineno = lineno;
                        $$->attr.name = savedName;
                        $$->type = Void;
                        }
            | VOID saveid LBRACE savenum RBRACE SEMI {
                        /*void array var*/
                        $$ = newExpNode(ArrVarK);
                        $$->lineno = lineno;
                        $$->attr.name = savedName;
                        $$->isArr = savedNumber;
                        $$->type = Void;
                        }
            ;
saveid      : ID { /*ID saving name string*/
                        savedName = copyString(tokenString);
                        savedLineNo = lineno;
                        }
            ;
savenum     : NUM { /*NUM saving constant number*/
                        savedNumber = atoi(tokenString);
                        savedLineNo = lineno;
                        }
            ;
fun_declaration: INT saveid { /*int func(){} declaration */
                        $$ = newExpNode(FuncK);
                        $$->lineno = lineno;
                        $$->attr.name = savedName;
                        $$->type = Integer;
                        } LPAREN params RPAREN compound_stmt {
                                    $$ = $3;
                                    $$->child[0] = $5;
                                    $$->child[1] = $7;
                        }
            | VOID saveid { /*void func(){} declaration */
                        $$ = newExpNode(FuncK);
                        $$->lineno = lineno;
                        $$->attr.name = savedName;
                        $$->type = Void;
                        } LPAREN params RPAREN compound_stmt {
                                    $$ = $3;
                                    $$->child[0] = $5;
                                    $$->child[1] = $7;
                        }
            ;
params      : param_list {$$ = $1;}/*one or more parameters*/
            | VOID {/*No Parameter (void)*/
                        $$ = newExpNode(ParamK);
                        $$->type = Void;
                        $$->isParam = TRUE;
                        }
            ;
```

(2)

```
param_list  : param_list COMMA param {
                        YYSTYPE t = $1;
                        if(t != NULL){
                          while(t->sibling != NULL){
                            t = t->sibling;
                          }
                          t->sibling = $3;
                          $$ = $1;
                        }
                        else{ $$ = $3;}
                        }
            | param {$$ = $1;}
            ;
param       : INT saveid { /*int var as a parameter*/
                         $$ = newExpNode(ParamK);
                        $$->attr.name = savedName;
                        $$->isParam = TRUE;
                        $$->type = Integer;
                        }
            | INT saveid LBRACE RBRACE {
                        /*int var[] as a parameter*/
                        $$ = newExpNode(ArrParamK);
                        $$->attr.name = savedName;
                        $$->isParam =TRUE;
                        $$->type = Integer;
                        }
            | VOID saveid { /*void var as a parameter*/
                         $$ = newExpNode(ParamK);
                        $$->attr.name = savedName;
                        $$->isParam = TRUE;
                        $$->type = Void;
                        }
            | VOID saveid LBRACE RBRACE {
                        /*void var[] as a parameter*/
                        $$ = newExpNode(ArrParamK);
                        $$->attr.name = savedName;
                        $$->isParam =TRUE;
                        $$->type = Void;
                        }
            ;
compound_stmt: LCURLY local_declarations statement_list RCURLY {
                        $$ = newStmtNode(CompndK);
                        $$->child[0] = $2;
                        $$->child[1] = $3;
                        }
            ;
local_declarations: local_declarations var_declaration {
                        YYSTYPE t = $1;
                        if(t != NULL){
                          while(t->sibling != NULL){
                            t = t->sibling;
                          }
                          t->sibling = $2;
                          $$ = $1;
                        }
                        else{$$ = $2;}
                        }
                        | {$$ = NULL;}
                        ;
statement_list : statement_list statement {
                        YYSTYPE t = $1;
                        if(t != NULL){
                          while(t->sibling != NULL){
                            t = t->sibling;
                          }
                          t->sibling = $2;
                          $$ = $1;
                        }
                        else{ $$ = $2; }
                        }
                        | {$$ = NULL;}
                        ;
statement   : expression_stmt {$$ = $1;}
            | compound_stmt {$$ = $1;}
            | selection_stmt {$$ = $1;}
            | iteration_stmt {$$ = $1;}
            | return_stmt {$$ = $1;}
            ;
expression_stmt         : expression SEMI {$$ = $1;}
                        | SEMI {$$ = NULL;}
                        ;
selection_stmt: IF LPAREN expression RPAREN statement
                    %prec LOWER_THAN_ELSE{
/*using %prec LOWER_THAN_ELSE to solve dangling else problem*/
                        $$= newStmtNode(SelectK);
                        $$->child[0] = $3;
                        $$->child[1] = $5;
                        $$->child[2] = NULL;
                        }
            | IF LPAREN expression RPAREN statement
              ELSE statement {
                        $$= newStmtNode(SelectK);
                        $$->child[0] = $3;
                        $$->child[1] = $5;
                        $$->child[2] = $7;
                        }
            ;
```

-<편집된 cminus.y (3),(4)>

(3)

```
iteration_stmt: WHILE LPAREN expression RPAREN statement {
                        $$= newStmtNode(IterK);
                        $$->child[0] = $3;
                        $$->child[1] = $5;
                        }
            ;
return_stmt : RETURN SEMI {
                        $$ = newStmtNode(ReturnK);
                        $$->child[0] = NULL;
                        }
            | RETURN expression SEMI {
                        $$ = newStmtNode(ReturnK);
                        $$->child[0] = $2;
                        }
            ;
expression  : var ASSIGN expression {
                        $$ = newStmtNode(AssignK);
                        $$->child[0] = $1;
                        $$->child[1] = $3;
                        }
            | simple_expression {$$ = $1;}
            ;
var         : saveid {
                        $$ = newExpNode(IdK);
                        $$->attr.name = savedName;
                        }
            | saveid {
                        $$ = newExpNode(IdK);
                        $$->attr.name = savedName;
                        } LBRACE expression RBRACE {
                                    $$ = $2;
                                    $$->child[0] = $4;
                                    }
            ;
simple_expression: additive_expression LE additive_expression {
                        $$ = newExpNode(RelOpK);
                        $$->child[0] = $1;
                        $$->child[1] = $3;
                        $$->attr.op = LE;
                        }
            | additive_expression LT additive_expression {
                        $$ = newExpNode(RelOpK);
                        $$->child[0] = $1;
                        $$->child[1] = $3;
                        $$->attr.op = LT;
                        }
            | additive_expression GT additive_expression {
                        $$ = newExpNode(RelOpK);
                        $$->child[0] = $1;
                        $$->child[1] = $3;
                        $$->attr.op = GT;
                        }
            | additive_expression GE additive_expression {
                        $$ = newExpNode(RelOpK);
                        $$->child[0] = $1;
                        $$->child[1] = $3;
                        $$->attr.op = GE;
                        }
            | additive_expression EQ additive_expression {
                        $$ = newExpNode(RelOpK);
                        $$->child[0] = $1;
                        $$->child[1] = $3;
                        $$->attr.op = EQ;
                        }
            | additive_expression NE additive_expression {
                        $$ = newExpNode(RelOpK);
                        $$->child[0] = $1;
                        $$->child[1] = $3;
                        $$->attr.op = NE;
                        }
            | additive_expression { $$ = $1; }
            ;
```

(4)

```
additive_expression: additive_expression PLUS term {
                        $$ = newExpNode(OpK);
                        $$->child[0] = $1;
                        $$->child[1] = $3;
                        $$->attr.op = PLUS;
                        }
            | additive_expression MINUS term {
                        $$ = newExpNode(OpK);
                        $$->child[0] = $1;
                        $$->child[1] = $3;
                        $$->attr.op = MINUS;
                        }
            | term {$$ = $1;}
            ;
term        : term TIMES factor {
                        $$ = newExpNode(OpK);
                        $$->child[0] = $1;
                        $$->child[1] = $3;
                        $$->attr.op = TIMES;
                        }
            | term OVER factor {
                        $$ = newExpNode(OpK);
                        $$->child[0] = $1;
                        $$->child[1] = $3;
                        $$->attr.op = OVER;
                        }
            | factor {$$ = $1;}
            ;
factor      : LPAREN expression RPAREN {$$ = $2;}
            | var {$$ = $1;}
            | call {$$ = $1;}
            | NUM {$$ = newExpNode(ConstK);
                        $$->attr.val = atoi(tokenString);}
            ;
call        : saveid {
                        $$ = newStmtNode(CallK);
                        $$->attr.name = savedName;
                        } LPAREN args RPAREN {
                                    $$ = $2;
                                    $$->child[0] = $4;
                        }
            ;
args        : arg_list {$$ = $1;}
            | {$$ = NULL;}
            ;
arg_list    : arg_list COMMA expression {
                        YYSTYPE t = $1;
                        if(t!=NULL){
                          while(t->sibling != NULL){
                            t = t->sibling;
                          }
                          t->sibling = $3;
                          $$ = $1;
                        }
                        else{ $$ = $3;}
                        }
            | expression {$$ = $1;}
            ;
%%
. . .
```

위의 코드에서 보라색으로 표시된 부분이 변경된 CFG에서 중요한 부분이다. 앞서 언급한 대로 op, relop, type_specifier 등의 non-terminal을 사용하지 않고, 직접 PLUS, TIMES, LE, INT 등의 terminal을 사용하여 treeNode를 생성함에 있어서 효과적으로 생성할 수 있게 바꾸어 주었고, ID 와 NUM의 문자열과 숫자를 저장할 수 있도록 saveid 와 savenum non-terminal을 따로 추가해 주었다. 또한 가장 큰 문제인 Dangling Else 문제를 해결하기 위해 "%nonassoc LOWER_THAN_ELSE"를 추가해주고 CFG production에서 "%prec LOWER_THAN_ELSE"를 추가해주어 shift/reduce conflict가 발생하는 것을 해결해 주었다.

(3) Edit "loucomp/util.c" in order to print appropriate Parsing Tree for CMINUS.

"util.c"에서는 newExpNode() 함수와 printTree() 함수를 편집해주었다. newExpNode() 함수에서 새로운 treeNode 를 생성할때, treeNode structure 에 추가된 변수인 isParam 과 isArr 변수를 초기화 해주는 부분을 추가 하였고, printTree()의 경우 올바른 Parsing Tree 를 잘 출력할 수 있도록 많은 부분을 편집해주었다.

-<편집된 util.c (1),(2)>

(1)

```
. . .
. . .
. . .

TreeNode * newExpNode(ExpKind kind)
{ TreeNode * t = (TreeNode *) malloc(sizeof(TreeNode));
  int i;
  if (t==NULL)
    fprintf(listing,"Out of memory error at line %d\n",lineno);
  else {
    for (i=0;i<MAXCHILDREN;i++) t->child[i] = NULL;
    t->sibling = NULL;
    t->nodekind = ExpK;
    t->kind.exp = kind;
    t->lineno = lineno;
    t->type = Void;
    t->isParam = FALSE;
    t->isArr = 0;
  }
  return t;
}


. . .
. . .
. . .


void printTree( TreeNode * tree )
{ int i;
  INDENT;
  while (tree != NULL) {
    printSpaces();
    if (tree->nodekind==StmtK)
    { switch (tree->kind.stmt) {
        case IfK://ignore
          fprintf(listing,"If\n");
         break;
        case RepeatK://ignore
          fprintf(listing,"Repeat\n");
          break;
        case AssignK:
          fprintf(listing,"Assign: (Destination) (Source)\n");
          break;
        case ReadK://ignore
          fprintf(listing,"Read: %s\n",tree->attr.name);
          break;
        case WriteK://ignore
          fprintf(listing,"Write\n");
          break;
case CompndK:
            fprintf(listing, "Compound statement: \n");
            break;
case SelectK:
  if(tree->child[2]!=NULL)
    fprintf(listing, "If: (Condition) (Body) (Else)\n");
  else
    fprintf(listing, "If: (Condition) (Body)\n");
  break;
case IterK:
  fprintf(listing, "while: (Condition) (Body)\n");
  break;
case ReturnK:
  fprintf(listing, "Return: \n");
  break;
case CallK:
  fprintf(listing, "Call, name: %s, with arguments below\n",tree-
>attr.name);
  break;
        default:
          fprintf(listing,"Unknown ExpNode kind\n");
          break;
      }
    }
    else if (tree->nodekind==ExpK)
    { switch (tree->kind.exp) {
        case OpK:
          fprintf(listing,"Op: ");
          printToken(tree->attr.op,"\0");
          break;
        case ConstK:
          fprintf(listing,"Const: %d\n",tree->attr.val);
          break;
        case IdK:
          fprintf(listing,"Id: %s\n",tree->attr.name);
          break;
```

(2)

```
case VarK:
  fprintf(listing, "Var Declaration, name: %s,",tree->attr.name);
  switch(tree->type){
    case Integer:
      fprintf(listing, " type: int\n");
      break;
    case Void:
      fprintf(listing, " type: void\n");
      break;
    default:
      fprintf(listing, "\n");
      break;
   }
  break;
case ArrVarK:
  fprintf(listing,"Array    Var    Declaration,    name:    %s,
size: %d,",tree->attr.name,tree->isArr);
  switch(tree->type){
    case Integer:
      fprintf(listing, " type: int array\n");
      break;
    case Void:
      fprintf(listing, " type: void array\n");
      break;
    default:
      fprintf(listing, "\n");
      break;
  }
  break;
case FuncK:
  fprintf(listing, "Function Declaration, name: %s, return",tree-
>attr.name);
  switch(tree->type){
    case Integer:
      fprintf(listing, " type: int\n");
      break;
    case Void:
      fprintf(listing, " type: void\n");
      break;
    default:
      fprintf(listing, "\n");
      break;
  }
  break;
case ArrParamK:
  fprintf(listing, "Array Parameter, name: %s,",tree->attr.name);
  switch(tree->type){
    case Integer:
      fprintf(listing, " type: int array\n");
      break;
    case Void:
      fprintf(listing, " type: void array\n");
      break;
    default:
      fprintf(listing, "\n");
      break;
  }
  break;
case ParamK:
  fprintf(listing,"Single Parameter, name: %s,",tree->attr.name);
  switch(tree->type){
    case Integer:
      fprintf(listing, " type: int\n");
      break;
    case Void:
      fprintf(listing, " type: void\n");
      break;
    default:
      fprintf(listing, "\n");
    break;
  }
  break;
case RelOpK:
  fprintf(listing, "RelOp: ");
  printToken(tree->attr.op,"\0");
  break;
       default:
         fprintf(listing,"Unknown ExpNode kind\n");
         break;
      }
    }
    else fprintf(listing,"Unknown node kind\n");
    for (i=0;i<MAXCHILDREN;i++)
        printTree(tree->child[i]);
    tree = tree->sibling;
  }
  UNINDENT;
}
```

위의 코드에서 보라색으로 표시된 부분이 Tree를 알맞게 출력 해주기 위해서 변경된 부분이다.

처음에는 expKind 에 TypeK 를 추가해서 type(int, void)정보를 child Node 에 저장하여 tree를 만들었는데, 이렇게 하면 printTree를 해줄 때 type정보가 한 줄에 출력되지 않고 밑에 줄에 출력이 되어서 이부분을 고치려 "cminus.y"에서 type_specifier를 없애고 생성된 Node에 직접 type정보를 추가하여 주는 방식으로 구현하여 과제 명세에 나와있는 대로 올바르게 잘 출력이 되었다. 또한 IF-ELSE를 출력할 때도 ELSE가 없는 IF 문과 ELSE가 존재하는 IF문을 구분하여 출력해주는 부분도 추가해 주었다.

또한 util.c를 편집하면서 기존의 tiny 언어에서만 사용하던 ReadK, WriteK와 같은 것을 모두 없애려 하였으나, 단순히 util.c에서만 없애게 되면 loucomp 내에 다른 모든 소스코드 파일과 같이 컴파일을 할 때 이것 때문에 오류가 발생하게 되어, 없애지 못하고 그냥 남겨두었다. (실제로 repeat 나 read 와 같은 tiny 에서의 reserved word를 변수로 사용하는 test case를 parsing 할 경우 cminus의 syntax에 관련하여서는 아무 문제가 없지만 syntax error를 출력한다. – 이 부분은 어쩔 수 없는 부분인 것 같다.)

(4) Edit "loucomp/main.c" , "Makefile" to generate Parse-Only CMINUS Parser executable file named "cimuns".

main.c 와 Makefile은 과제 명세에 나와있는 그대로 편집해 주었다.

Results:

여러 경우의 수를 확인해보기 위해 기존의 test.cm 파일을 조금 편집하여 여러가지 testcase를 만들어 실행해보았다.

(1) Default "test.cm" & result of execution.

test.cm (~/2019compiler/2019_

```
1  /*A program to perform Euclid's
2    Algorithm to compute gcd*/
3
4  int gcd(int u, int v)
5  {
6      if(v==0) return u;
7      else return gcd(v,u-u/v*v);
8      /* u-u/v*v == u mod v */
9  }
10
11 void main(void)
12 {
13     int x; int y;
14     x = input(); y = input();
15     output(gcd(x,y));
16 }
```

sangheon@sangheon-VirtualBox: ~/2019compiler/201

```
sangheon@sangheon-VirtualBox:~/2019compiler/2019_ELE4029_2
p$ ./cminus test.cm

C-MINUS COMPILATION: test.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)
        RelOp: ==
          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
```

- 기본 test.cm을 실행시켜 봤을 때, 과제 명세에 나온 결과와 같은 결과를 얻을 수 있었다.
- (Parsing Tree를 Print 해줄때, 비교연산과, 일반 연산을 구분하여, 비교연산은 Relop, 일반연산은 Op 로 구분하여 출력해주도록 하였다.)

(2) "testtest.cm" & result of execution. (for Dangling Else)

testtest.cm + (~/2019compiler/201

```
1 void main(void){
2     if(i==0) if(j==0) d=1; else d=2;
3 }
```

- testtest.cm은 Dangling IF ELSE 를 잘 해결하여 출력하는지를 확인하기 위한 testcase 이다.
- 결과를 보면 else가 가장 가까운 if와 sibling이 되어있는 것을 확인할 수 있고, 알맞게 구현되었다는것을 확인 할 수 있다.

sangheon@sangheon-VirtualBox: ~/2019compiler/201

```
sangheon@sangheon-VirtualBox:~/2019compiler/2019_ELE4029_2
p$ ./cminus testtest.cm

C-MINUS COMPILATION: testtest.cm

Syntax tree:
  Function Declaration, name: main, return type: void
    Single Parameter, name: (null), type: void
    Compound statement:
      If: (Condition) (Body)
        RelOp: ==
          Id: i
          Const: 0
        If: (Condition) (Body) (Else)
          RelOp: ==
            Id: j
            Const: 0
          Assign: (Destination) (Source)
            Id: d
            Const: 1
          Assign: (Destination) (Source)
            Id: d
            Const: 2
```

(3) "test02.cm" & result of execution. (for Array Parameter, Array Variable)

test02.cm (~/2019compiler/2019_E

```
1  /*A program to perform Euclid's
2    Algorithm to compute gcd*/
3
4  int gcd(int u[], int v)
5  {
6      if(v==0) return u[0];
7      else return gcd(v,u[0]-u[0]/v*v);
8      /* u-u/v*v == u mod v */
9  }
10
11 void main(void)
12 {
13     int x; int y;
14     x = input(); y = input();
15     if(x) x = x+1;
16     output(gcd(x,y));
17 }
```

- test02.cm은 Array Parameter와 Array Variable 들을 잘 출력하는지를 확인하기 위한 testcase 이다.
- 결과를 보면 Array Parameter를 잘 출력해주고, Array Variable에 index를 이용하여 접근할때, index를 child노드로 갖고 있어서 child로 출력해주는 것을 확인 할 수 있기에 잘 구현 되었다고 생각한다.

u[0] 를 표현할때
Id : u
Const : 0
로 표현한다. (index는 child Node에 들어간다.)

sangheon@sangheon-VirtualBox: ~/2019compiler/201

```
sangheon@sangheon-VirtualBox:~/2019compiler/2019_ELE4029_2
p$ ./cminus test02.cm

C-MINUS COMPILATION: test02.cm

Syntax tree:
  Function Declaration, name: gcd, return type: int
    Array Parameter, name: u, type: int array
    Single Parameter, name: v, type: int
    Compound statement:
      If: (Condition) (Body) (Else)
        RelOp: ==
          Id: v
          Const: 0
        Return:
          Id: u
            Const: 0
        Return:
          Call, name: gcd, with arguments below
            Id: v
            Op: -
              Id: u
                Const: 0
              Op: *
                Op: /
                  Id: u
                    Const: 0
                  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
      If: (Condition) (Body)
        Id: x
        Assign: (Destination) (Source)
          Id: x
          Op: +
            Id: x
            Const: 1
      Call, name: output, with arguments below
        Call, name: gcd, with arguments below
          Id: x
          Id: y
```

(4) "test2.cm" & result of execution. (for Array Variable declaration & Array indexing)

test2.cm (~/2019compiler/

```
1  void main(void)
2  {
3      int i; int x[5];
4
5      i = 0;
6      while( i < 5 )
7      {
8          x[i] = input();
9
10         i = i + 1;
11     }
12
13     i = 0;
14     while( i <= 4 )
15     {
16         if( x[i] != 0 )
17         {
18             output(x[i]);
19         }
20     }
21 }
```

- test2.cm은 Array variable Declaration과 Array indexing이 잘 출력되는지를 확인하기 위한 testcase 이다.
- 결과를 보면 int array type과 배열의 size를 잘 출력해주고, Array Variable에 index를 이용하여 접근할때, index를 child 노드로 갖고 있어서 child로 출력해주는 것을 확인 할 수 있기에 잘 구현 되었다고 생각한다.

sangheon@sangheon-VirtualBox: ~/2019compiler/2019_ELE

```
sangheon@sangheon-VirtualBox:~/2019compiler/2019_ELE4029_201500
p$ ./cminus test2.cm

C-MINUS COMPILATION: test2.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 array
      Assign: (Destination) (Source)
        Id: i
        Const: 0
      while: (Condition) (Body)
        RelOp: <
          Id: i
          Const: 5
        Compound statement:
          Assign: (Destination) (Source)
            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)
        RelOp: <=
          Id: i
          Const: 4
        Compound statement:
          If: (Condition) (Body)
            RelOp: !=
              Id: x
                Id: i
              Const: 0
            Compound statement:
              Call, name: output, with arguments below
                Id: x
                  Id: i
sangheon@sangheon-VirtualBox:~/2019compiler/2019_ELE4029_201500
```

/*Project02 에선 보고서의 분량 제한에 대한 명시가 없어서 5장을 넘겨도 되는 걸로 이해하고 8장으로 제출합니다*/