Project #2. Scanner

컴퓨터소프트웨어학부 2017029589 류지범

#0 Environment & Execute

- Environment
  - Ubuntu 18.04.02 LTS
- Execute
  - Makefile로 컴파일한 후 다음과 같이 실행한다.
  - `./cminus_parser input.txt`

#1 기본 세팅

- main.c
  - `NO_PARSE` 를 `FALSE`로 고치고 `NO_ANALYZE`를 `TRUE`로 고쳤다.
- globals.h
  - Declaration을 위한 `DeclKind`를 추가해주었다.
  - cminus 문법에 맞는 노드를 생성하기 위해 `StmtKind`, `ExpKind`, `DeclKind`를 재정의해주었다.

```
typedef enum {StmtK, ExpK, DeclK} NodeKind;
typedef enum {CompK, IfK, WhileK, ReturnK} StmtKind;
typedef enum {AssignK, OpK, CallK, ConstK, IdK, ArrayK} ExpKind;
typedef enum {VarK, FuncK, ParamK} DeclKind;
```

  - `treeNode`의 `kind`에 `DeclKind decl;`을 추가해줬다.
- util.c
  - Declaration을 나타내기 위한 declaration Node를 만들기 위해 다음 함수를 추가해줬다.

```
TreeNode * newDeclNode(DeclKind 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 = DeclK;
        t->kind.decl = kind;
        t->lineno = lineno;
    }
    return t;
}
```

#2 cminus.y (BNF)

- cminus.y의 BNF를 통해 Parsing table이 만들어지고 cminus 문법에 맞도록 Parsing을 할 수 있게 된다.
- BNF는 Appendix A.2의 문법을 따랐다.
- 1_Scanner에서 사용했던 token들을 똑같이 정의해주었다.

```
%token IF ELSE WHILE RETURN INT VOID
%token ID NUM
%token ASSIGN EQ LT LE GT GE NE PLUS MINUS TIMES OVER LPAREN RPAREN LBRACE
RBRACE LCURLY RCURLY SEMI COMMA
%token ERROR
```

- Left recursion 형태

```
decl_list   : decl_list decl
                { YYSTYPE t = $1;
                  if (t != NULL)
                  { while (t->sibling != NULL)
                      t = t->sibling;
                    t->sibling = $2;
                    $$ = $1; }
                  else $$ = $2;
                }
            | decl { $$ = $1; }
            ;
```

  - A -> AB 꼴의 Left recursion의 경우 sibling이 NULL이 아닐 때까지 계속 탐색하고, 마지막 sibling에게 B를 할당해주도록 구현했다.
  - 이러한 형태를 따르는 Grammar rule은 `param_list`, `local_decl`, `stmt_list`, `arg_list`가 있다.

- ID를 포함한 Rule의 경우
  - 우선 identifier는 savedName에 저장되어야하므로, ID에 포함된 rule에는 `savedName = copyString(tokenString);`이 포함되어야 하므로, 이 과정을 `id      : ID { savedName = copyString(tokenString); }`라는 rule을 만들어 처리하도록 했다.
  - savedName에 Identifier를 저장해서 이후에 노드의 name attr에 할당해주도록 하는데, 이때 ID 뒤에 여러 문법이 이어져있을 경우 SavedName이 우리가 의도한 Identifier로 지정되지 않는다. SavedName은 가장 최근의 Identifier를 저장하고 있기 때문이다.
  - 그래서 ID를 포함한 rule의 경우 non-terminal이 나오기 전에 attr.name에 savedName을 할당해주는 action을 취하도록 했다.
  - 다음과 같은 rule들이 이에 해당한다. 항상 action은 non-terminal이 나오기 전에 취해준다.

```
var_decl    : type_spec id SEMI
                  { $$ = newDeclNode(VarK);
```

```
                    $$->attr.name = savedName;
                    $$->type = $1->type;
                  }
              | type_spec
                id
                    { $$ = newDeclNode(VarK);
                      $$->attr.name = savedName;
                      $$->type = $1->type;
                    }
                LBRACE
                NUM {
                    $$ = $3;
                    $$->child[0] = newExpNode(ConstK);
                    $$->child[0]->attr.val = atoi(tokenString); }
                RBRACE SEMI { $$ = $6; }
              ;
func_decl     : type_spec id
                    { $$ = newDeclNode(FuncK);
                      $$->attr.name = savedName; }
                LPAREN params RPAREN comp_stmt
                    { $$ = $3;
                      $$->child[0] = $5;
                      $$->child[1] = $7;
                      $$->type = $1->type;
                    }
              ;
```

- 각 rule들은 ppt의 node 양식을 따랐다.
  - variable declaration은 declNode로 할당했고 VarKind를 가지도록 했다.
    - name, type을 가지고 있고, array인 경우 array size를 child[0]으로 가지고 있도록 했다.
  - function declaration은 declNode로 할당했고 FuncKind를 가지도록 했다.
    - name, type을 가지고 있고, child[0]으로 params, child[1]으로 compound_stmt를 가지고 있다.
  - parameter는 declNode로 할당했고 ParamKind를 가지도록 했다.
    - name, type을 가지고 있고, void parameter일 경우 name은 NULL이다.
    - array일 경우 chlid[0]으로 ExpNode(ConstK)를 할당했고 const의 값인 attr.val은 -1로 할당했다.
  - Compound_stmt는 StmtNode로 할당했고 CompKind를 가지도록 했다.
    - child[0]으로 local_declaration, child[1]으로 statement_list를 가지고 있다.
  - Selection_stmt는 StmtNode로 할당했고 IfKind를 가지도록 했다.
    - child[0]으로 expression, child[1]으로 statement를 가지고 있거나, child[0]으로 expression, child[1]으로 statement, child[2]로 statement를 가지고 있다.
  - While_statement는 StmtNode로 할당했고 WhileKind를 가지도록 했다.
    - child[0]으로 expression, child[1]으로 statement를 가지고 있다.
  - return statement는 StmtNode로 할당했고 ReturnKind를 가지도록 했다.
    - `return;`의 경우 기본 노드로 생성되고 이외의 경우는 child[0]으로 expression을 가지고 있다.
  - expression은 ExpNode로 할당했고 AssignKind를 가지도록 했다.

        - child[0]으로 variable, child[1]로 expression을 가지고 있다.
    - variable은 ExpNode로 할당했다.
        - 변수인 경우 name을 가지고 있고 IdKind를 가지도록 했다.
        - array인 경우 child[0]으로 expression을 가지고 있고 ArrayKind를 가지도록 했다.
    - Simple_expression, addictive_expression, term의 경우 child[0]으로 lhs를 가지고 있고, child[1]로 rhs를 가지고 있다.
    - NUM의 경우 ExpNode로 할당했고 ConstKind를 가지도록 했다.
        - number를 가지고 있다.
    - Call_expression의 경우 ExpNode로 할당했고 CallKind를 가지도록 했다.
        - name을 가지고 있고 child[0]으로 args를 가지고 있다.
    - 연산자의 경우 ExpNode로 할당했고 OpKind를 가지도록 했다.
- Dangling Else Problem
    - if else문의 경우 Dangling Else Problem이 발생할 수 있다. 아래는 그 예시이다.

```
select_stmt : IF LPAREN exp RPAREN stmt
            | IF LPAREN exp RPAREN stmt ELSE stmt
```

    - 그래서 위의 BNF만으로 구현하면 Shift/Reduce conflict가 발생한다.
    - else가 token으로 들어왔을 때 else와 가장 가까운 if문을 수행할 수 있도록 해야한다.
    - 이는 else의 우선순위가 높음을 의미하고 token의 우선순위가 높기 때문에 이 경우 Shift를 취할 수 있도록 해준다.
    - 이러한 방식은 결국 right assocation을 뜻하기 때문에 %right로 THEN ELSE를 추가해서 right associativity를 따르도록 해주고 우선순위를 다음과 같이 바꾸어준다.

```
select_stmt : IF LPAREN exp RPAREN stmt %prec THEN
            | IF LPAREN exp RPAREN stmt ELSE stmt
```

- 연산자 우선순위의 경우 문법자체가 우선순위를 맞추어 구현되었기 때문에 따로 설정해주지 않았다.
- 에러 해결을 위해 `static int yylex(void);` `int yyerror(char *s);`을 맨 윗 부분에 추가해줬다.

#3 util.c (Syntax Tree 출력)

- `printTree`는 tiny의 원래 기본 양식을 따라서 작성했다.
- Declaration 을 출력하기 위해 Declaration Node에 해당하는 switch문을 추가해주었다.
- 각각의 노드와 kind에 맞는 출력형식으로 출력하도록 했다.
- Non-value return Statement
    - return 문의 경우 child[0]이 NULL일 경우 Non-value return statement를 출력하도록 했다.
- Void parameter
    - `int main(void)`의 경우 void parameter인데 이는 attr.name이 NULL일 경우 Void parameter로 출력하도록 했다.

- type 출력은 여러 군데에서 쓰는 공통된 작업이므로 하나의 함수로 만들어서 처리했다.

```
void printExpType(ExpType type) {
    switch(type) {
        case Void:
            fprintf(listing, "void");
            break;
        case Integer:
            fprintf(listing, "int");
            break;
        default:
            fprintf(listing, "error type");
            break;
    }
}
```

- type의 경우 array인 것이 paramter와 variable에서 출력되기 때문에 이것 역시 하나의 함수로 만들어서 처리했다.

```
void printDeclType(TreeNode * tree) {
    fprintf(listing, "type = ");
    printExpType(tree->type);
    if (tree->child[0] != NULL) {
        fprintf(listing, "[]\n");
    } else {
        fprintf(listing, "\n");
    }
}
```

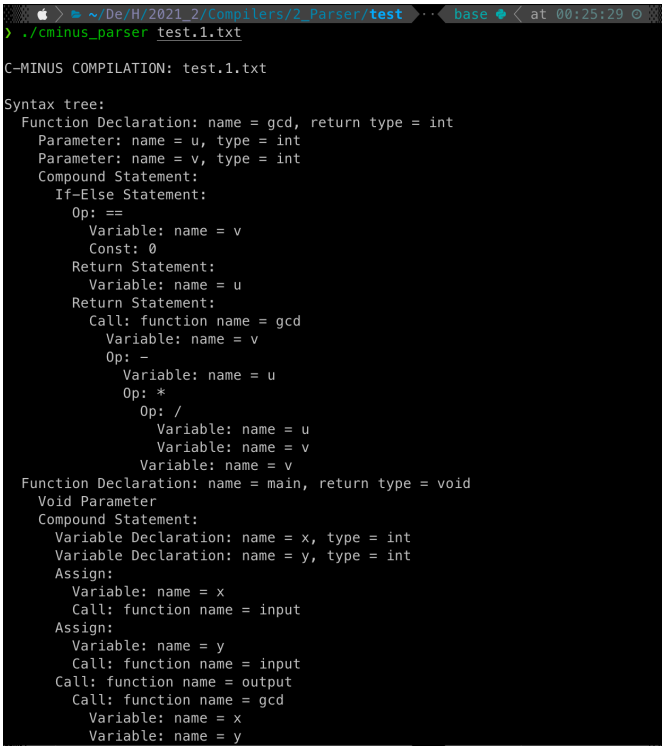
  - array일 경우 []를 추가로 출력하도록 했다.

- If-Else Statement
  - If문의 경우 child[2]가 NULL이 아닐 경우 If-Else Statement를 출력하도록 했다.
- Parameter의 경우 Array type인 경우 child를 가진다. 그리고 child의 attr.val의 값이 -1을 가지도록 설정했다.
  - 하지만 child를 출력하면 안되기 때문에 이 경우 check라는 변수를 0으로 설정하도록 한다.
  - check가 0일 경우에는 child를 출력하지 않도록 마지막 코드를 수정한다.

```
if (check) {
            for (i=0;i<MAXCHILDREN;i++)
                printTree(tree->child[i]);
        }
```

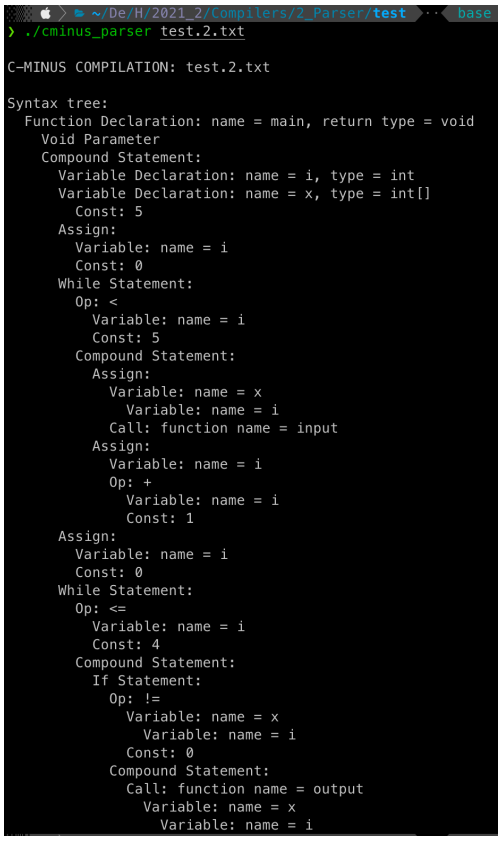
#4 Result

- test case 1, 2는 주어진 것을 사용했다.
  - test.1.txt



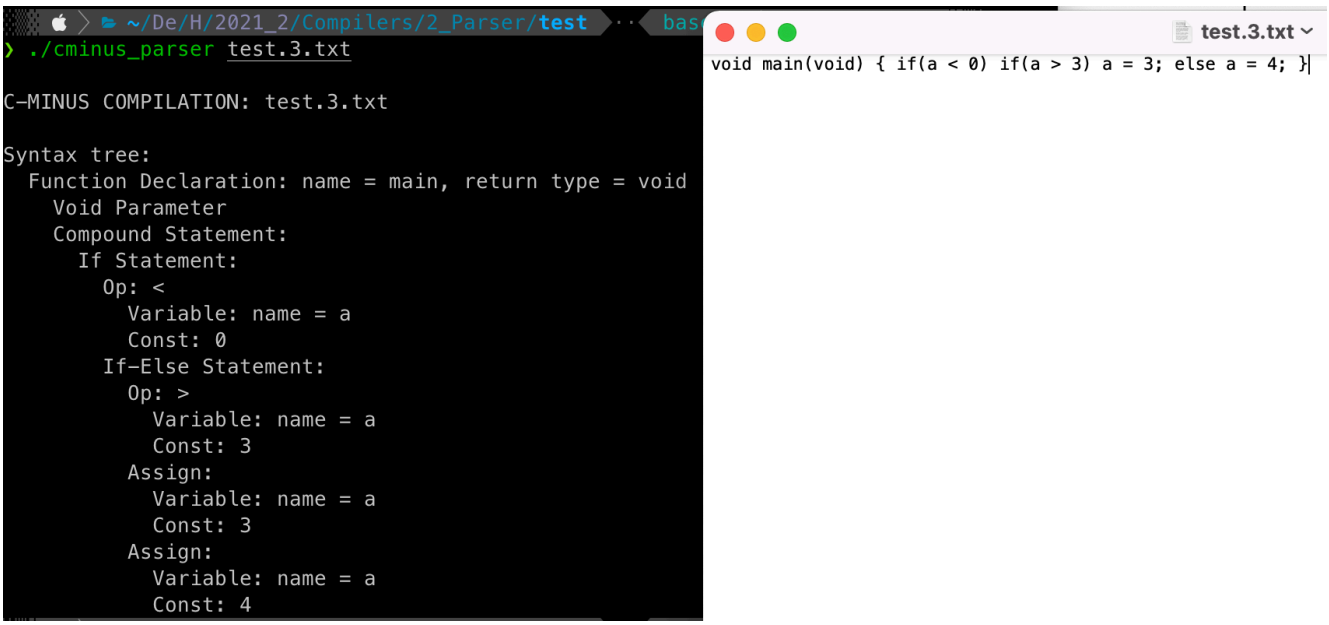
```
~/De/H/2021_2/Compilers/2_Parser/test  base  at 00:25:29
❯ ./cminus_parser test.1.txt

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

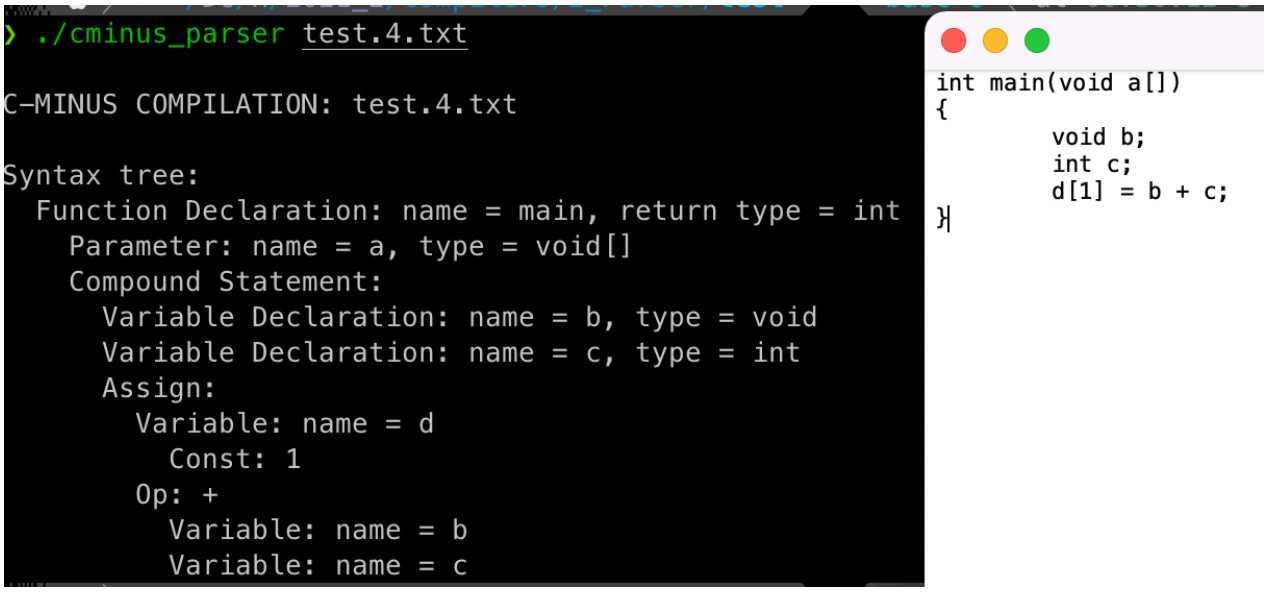


```
~/De/H/2021_2/Compilers/2_Parser/test  base
❯ ./cminus_parser test.2.txt

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 = x
              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 = x
                Variable: name = i
              Const: 0
            Compound Statement:
              Call: function name = output
                Variable: name = x
                  Variable: name = i
```

- Test case 3, 4는 ppt를 참고했고 5는 개인적으로 만들었다.
  - test.3.txt (Dangling Else Problem)



  - test.4.txt



- test.5.txt

```
C-MINUS COMPILATION: test.5.txt

Syntax tree:
  Function Declaration: name = max, return type = int
    Parameter: name = a, type = int
    Parameter: name = b, type = int
    Compound Statement:
      If-Else Statement:
        Op: >
          Variable: name = a
          Variable: name = b
        Return Statement:
          Variable: name = a
        Return Statement:
          Variable: name = b
  Function Declaration: name = main, return type = int
    Parameter: name = a, type = int[]
    Parameter: name = b, type = int
    Parameter: name = c, type = void[]
    Parameter: name = d, type = void
    Compound Statement:
      Variable Declaration: name = x, type = int[]
        Const: 4
      Variable Declaration: name = y, type = int
      Variable Declaration: name = z, type = int[]
        Const: 10
      Assign:
        Variable: name = z
          Const: 4
        Op: +
          Const: 5
          Op: *
            Const: 3
            Const: 4
      Assign:
        Variable: name = a
          Const: 1
        Op: +
          Variable: name = b
          Variable: name = d
      Assign:
        Variable: name = x
        Call: function name = max
          Call: function name = max
            Variable: name = x
            Variable: name = y
          Variable: name = z
            Const: 1
      While Statement:
        Op: <
          Variable: name = x
          Const: 5
        Compound Statement:
          Assign:
            Variable: name = x
            Op: +
              Op: /
                Variable: name = x
                Const: 2
              Const: 1
      Return Statement:
        Variable: name = z
          Const: 5
```

test.5.txt

```
int max(int a, int b)
{
        if(a > b) return a;
        else return b;
}

int main(int a[], int b, void c[], void d)
{
        int x[4]; int y;
        int z[10];
        z[4] = 5 + 3 * 4;
        a[1] = b + d;

        x = max(max(x, y), z[1]);
        while(x < 5)
        {
                x = x / 2 + 1;
        }

        return z[5];
}
```