Project #2. Parser

2018007938 김민관

Goal

- Yacc을 사용해서 C-Minus Parser 를 구현한다.
    - source code를 읽고, c-minus 문법을 통해 tokenize와 parser를 진행한 후, abstract syntax tree(AST)를 만든다.

Implement

1. main.c

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

/* allocate and set tracing flags */
int EchoSource = FALSE;
int TraceScan = FALSE;
int TraceParse = TRUE;
int TraceAnalyze = FALSE;
int TraceCode = FALSE;
```

main 함수에서 c-minus parser를 사용하기 위한 flag 값들을 조정한다.

2.globals.h

```
/**************************************************/
/***********   Syntax tree for parsing ************/
/**************************************************/
```

```
typedef enum {StmtK,ExpK,DeclareK,ParameterK,TypeK} NodeKind;
typedef enum {CompK,IfK,IfEK,IterK,RetK} StmtKind;
typedef enum {AssignK,OpK,ConstK,IdK,ArrIdK,CallK} ExpKind;
typedef enum {FunctionK,VariableK,ArrayVariableK} DeclareKind;
typedef enum {ArrayParameterK,NonArrayParameterK} ParameterKind;
typedef enum {TypeNameK} TypeKind;

/* For array tree*/
typedef struct arrayAttr
   { TokenType type;
     char * name;
     int size;
   } ArrayAttr;

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

#define MAXCHILDREN 3

typedef struct treeNode
   { struct treeNode * child[MAXCHILDREN];
     struct treeNode * sibling;
     int lineno;
     NodeKind nodekind;
     union { StmtKind stmt;
             ExpKind exp;
             DeclareKind decl;
             ParameterKind param;
             TypeKind type; } kind;
     union { TokenType op;
             TokenType type;
             int val;
             char * name;
             ArrayAttr arr; } attr;
     ExpType type; /* for type checking of exps */
   } TreeNode;
```

yacc파일에 있는 globals.h 파일을 복사하여 수정하는 방향으로 진행했다. NodeKind의 enum에는 DeclareK, ParameterK, TypeK를 추가하였고 각 NodeKind에 따라 enum의 값을 추가하였다.

DeclareKind 에서는 배열을 다루기위한 enum인 ArrayVariable을 추가했다. 그리고 ArrayAttr 구조체를 따로 구현하여 배열을 인식한도록 하고 이에 따라 treeNode구조체 또한 수정했다.

3.util.c

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


TreeNode * newParamNode(ParameterKind 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 = ParameterK;
    t->kind.param = kind;
    t->lineno = lineno;
  }
  return t;
}


TreeNode * newTypeNode(TypeKind 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 = TypeK;
    t->kind.type = kind;
    t->lineno = lineno;
  }
  return t;
}
```

newStmtNode(), newExprNode()외에도 BNF에서 Declare, Parameter, Type Node가 더 추가 되었으므로, newDeclNode, newParamNode, newTypeNode 함수도 추가해준다.

```
void printTree( TreeNode * tree )
{ int i;
  int returnType;
  int typeShow;
```

```
    INDENT;
    while (tree != NULL)
    { if (tree->nodekind!=TypeK)
        printSpaces();
      if (tree->nodekind==StmtK)
      { switch (tree->kind.stmt) {
          case CompK:
            fprintf(listing,"Compound statement :\n");
            break;
          case IfK:
            fprintf(listing,"If Statement\n");
            break;
          case IfEK:
            fprintf(listing,"If-Else Statement\n");
            break;
          case IterK:
            fprintf(listing,"While Statement : \n");
            break;
          case RetK:
            fprintf(listing,"Return Statement: \n");
            break;
          default:
            fprintf(listing,"Unknown ExpNode kind\n");
            break;
        }
      }
      else if (tree->nodekind==ExpK)
      { switch (tree->kind.exp) {
          case AssignK:
            fprintf(listing,"Assign : \n");
            break;
          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,"Variable : name = %s\n",tree->attr.name);
            break;
          case ArrIdK:
            fprintf(listing,"Variable : name : %s\n",tree->attr.name);
            break;
          case CallK:
            fprintf(listing,"Call : function name =  %s\n",tree->attr.name);
            break;
          default:
            fprintf(listing,"Unknown ExpNode kind\n");
            break;
        }
      }
      else if (tree->nodekind==DeclareK)
      { switch (tree->kind.decl) {
          case FunctionK:
            fprintf(listing,"Function declaration, name : %s, return ",tree->attr.name);
            break;
          case VariableK:
```

```
          fprintf(listing,"Variable Declaration, name : %s, ",tree->attr.name);
          break;
        case ArrayVariableK:
          fprintf(listing,"Variable Declaration, name : %s, ",tree->attr.arr.name);
          break;
        default:
          fprintf(listing,"Unknown DeclNode kind\n");
          break;
      }
    }
    else if (tree->nodekind==ParameterK)
    { switch (tree->kind.param) {
        case ArrayParameterK:
          fprintf(listing,"Array parameter, name : %s, ",tree->attr.name);
          break;
        case NonArrayParameterK:
          if(returnType == VOID)
          {
            fprintf(listing,"Void Parameter\n");
            typeShow = 0;
          }
          else fprintf(listing,"Parameter, name : %s, ",tree->attr.name);
          break;
        default:
          fprintf(listing,"Unknown ParamNode kind\n");
          break;
      }
    }
    else if (tree->nodekind==TypeK)
    { switch (tree->kind.type) {
        case TypeNameK:
        if(typeShow != 0)
        {
          fprintf(listing,"type : ");
          switch (tree->attr.type) {
            case INT:
              if(tree->kind.decl == ArrayVariableK)
              {
                returnType = INT;
                fprintf(listing, "int[]\n");
                break;
              }
              else
              {
                returnType = INT;
                fprintf(listing,"int\n");
                break;
              }
            case VOID:
              returnType = VOID;
              fprintf(listing,"void\n");
              break;
          }
        }
        break;
        default:
          fprintf(listing,"Unknown TypeNode kind\n");
          break;
```

```
      }
    }
    else fprintf(listing,"Unknown node kind\n");
    for (i=0;i<MAXCHILDREN;i++)
         printTree(tree->child[i]);
    tree = tree->sibling;
  }
  UNINDENT;
}
```

printTree() 함수를 통해 C-Minus Tree를 출력할 내용을 update한다. 출력 내용은 result1과 result2를 비교하여 내용을 기반으로 수정했다. tree->attr.type 값이 INT일 경우, 이를 그냥 int 일 값과 int[ ] 을 구분해서 관리한다. 또한 returnType 변수를 만들어 returnType이 VOID 일 경우와, INT일 경우를 나누어서 Tree를 print한다.

4. cminus.y

```
program     : stmt_seq
                { savedTree = $1;}
            ;
stmt_seq    : stmt_seq SEMI stmt
                { YYSTYPE t = $1;
                  if (t != NULL)
                  { while (t->sibling != NULL)
                       t = t->sibling;
                    t->sibling = $3;
                    $$ = $1; }
                    else $$ = $3;
                }
            | stmt  { $$ = $1; }
            ;
stmt        : if_stmt { $$ = $1; }
            | repeat_stmt { $$ = $1; }
            | assign_stmt { $$ = $1; }
            | read_stmt { $$ = $1; }
            | write_stmt { $$ = $1; }
            | error  { $$ = NULL; }
            ;

  .
  .
  .
  .

factor      : LPAREN exp RPAREN
                { $$ = $2; }
            | NUM
                { $$ = newExpNode(ConstK);
                  $$->attr.val = atoi(tokenString);
                }
```

```
            | ID { $$ = newExpNode(IdK);
                   $$->attr.name =
                         copyString(tokenString);
                 }
            | error { $$ = NULL; }
            ;

%%

int yyerror(char * message)
{ fprintf(listing,"Syntax error at line %d: %s\n",lineno,message);
  fprintf(listing,"Current token: ");
  printToken(yychar,tokenString);
  Error = TRUE;
  return 0;
}

/* yylex calls getToken to make Yacc/Bison output
 * compatible with ealier versions of the TINY scanner
 */
static int yylex(void)
{ return getToken(); }

TreeNode * parse(void)
{ yyparse();
  return savedTree;
}
```

cminus.y 또한 yacc파일의 tiny.y를 복사하여 수정하는 방향으로 진행했다.

```
var_decl    : type_spec saveName SEMI
                 { $$ = newDeclNode(VariableK);
                   $$->child[0] = $1;
                   $$->lineno = lineno;
                   $$->attr.name = savedName;
                 }
            | type_spec saveName LBRACE saveNumber RBRACE SEMI
                 { $$ = newDeclNode(ArrayVariableK);
                   $$->child[0] = $1;
                   $$->lineno = lineno;
                   $$->attr.arr.name = savedName;
                   $$->attr.arr.size = savedNumber;
                 }
            ;

var         : saveName
                 { $$ = newExpNode(IdK);
                   $$->attr.name = savedName;
                 }
            | saveName
                 { $$ = newExpNode(ArrIdK);
                   $$->attr.name = savedName;
                 }
```

```
                LBRACE exp RBRACE
                   { $$ = $2;
                     $$->child[0] = $4;
                   }
             ;
```

var_decl과 var 모두 newExpNode(ArrIdk)를 추가하여 배열의 관련된 내용을 추가한다.

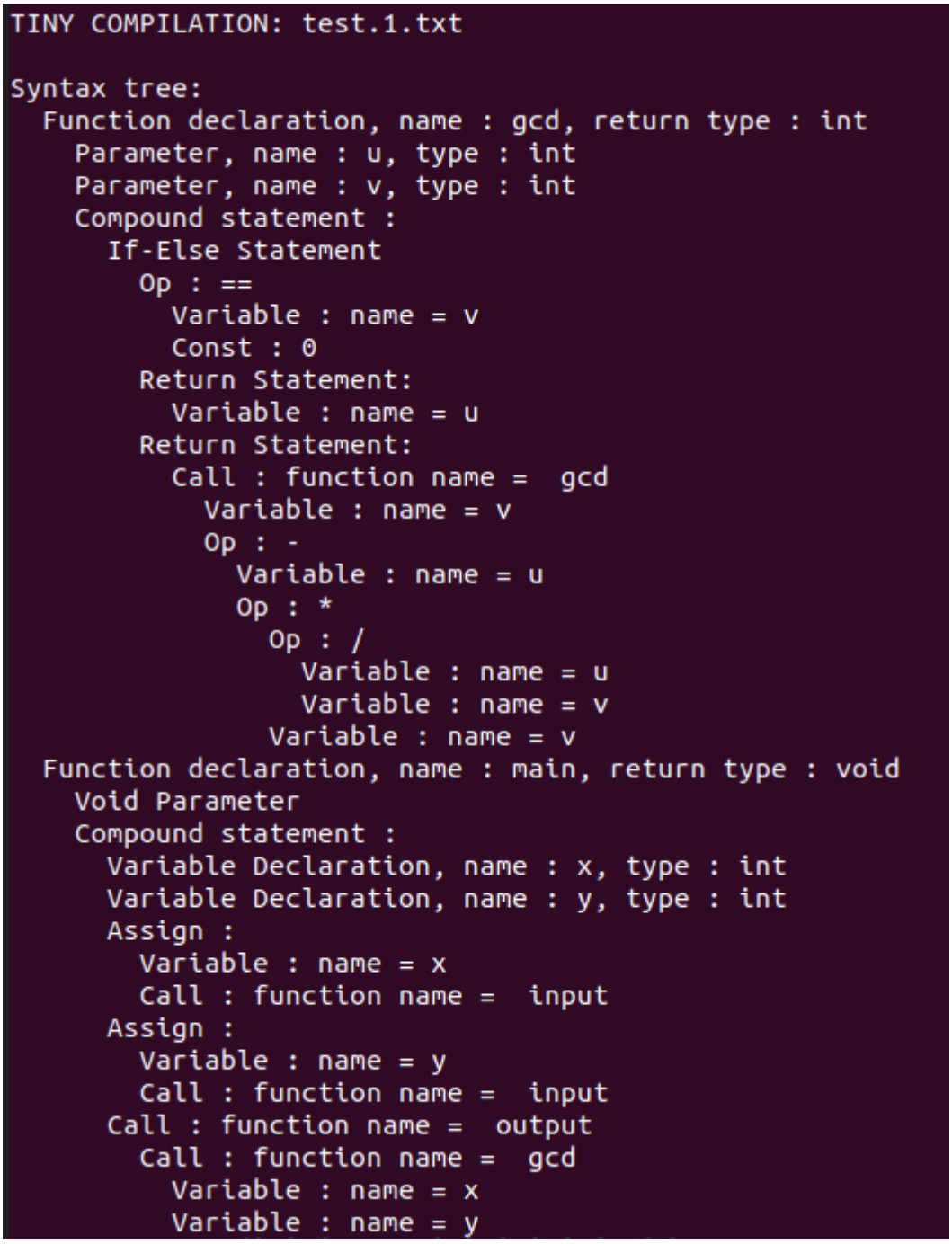
5. MakeFile

MakeFile의 경우 제공된걸 사용하였고, make를 진행할 시, lex.yy.c, y.tab.c, y.tab.h 가 새로 생성된다.

Result

```
make cminus_parser
./cminus_parser test.1.txt
```

위의 명령어를 terminal에 입력하여 make하고 test.1.txt 파일의 내용을 ast 로만들면 아래와 같은 결과가 나타난다.

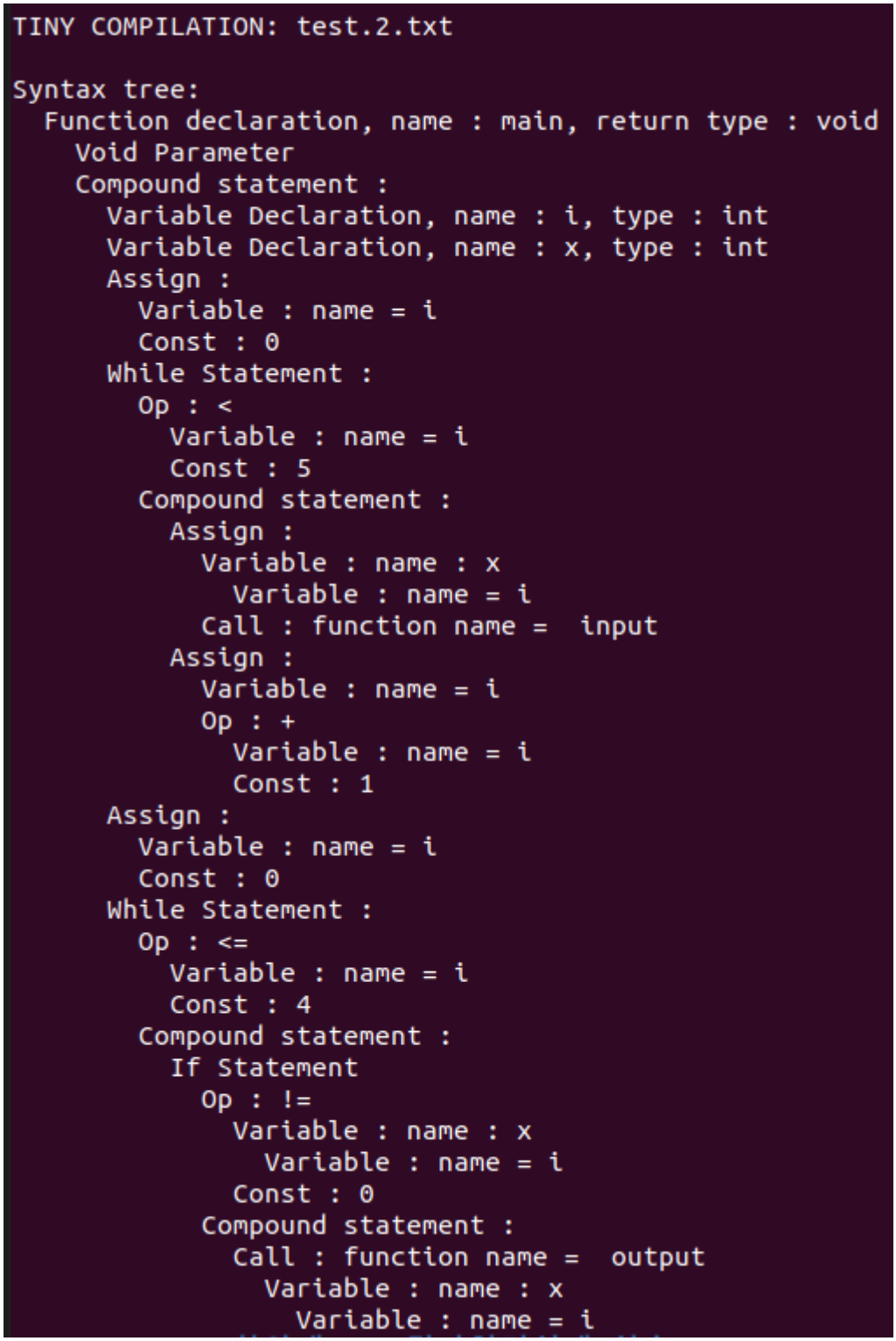


```
TINY COMPILATION: test.1.txt

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
```

```
./cminus_parser test.2.txt
```

test.2.txt 파일을 ast로 만들면 아래와 같은 결과 값이 나온다.



```
TINY 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
      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
```