Project1 Scanner

2019094511 김준표

1. Compilation method and environment

- Environment: Windows 10 WSL2 – Ubuntu 18.04

- Run:

make → 컴파일 실시, cminus_parser 생성

execute → ./cminus_parser test.1.txt

2. Explanation about how to implement and how to operate + Code

- globals.h

cminus.y에서 사용되는 노드를 정의한다. 해당부분에서는 노드의 종류와 tree노드 구조체가 정의되어 있으며 이후 사용될 변수들을 추가하고 목적에 맞게 묶어줬다.

```
typedef enum {DeclK,StmtK,ExpK} NodeKind;
typedef enum {VarK,ArrVarK,FunK,ParamK,ArrParamK,TypeK} DeclKind;
typedef enum {CompK,IfK,WhileK,ReturnK} StmtKind;
typedef enum {AssignK,OpK,ConstK,IdK,IdxK,CallK} 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 { DeclKind decl; StmtKind stmt; ExpKind exp;} kind;
     union { TokenType op;
             int val;
             int size;
             char * name;
             char * type; } attr;
     ExpType type; /* for type checking of exps */
   } TreeNode;
```

- util.c

선언을 위한 DeclNode를 이용하기 위해 newDeclNode()함수를 추가해줬으며 이의 구성은 기존의 newStmtNode(), newExpNode()와 동일하다.

```c
/* Function newDeclNode creates a new declaration
 * node for syntax tree construction
 */
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;
}
```

printTree함수

해당 함수에서는 parsing된 결과값을 출력하는 함수로 cminus.y의 결과를 받아와 출력한다. 해당 부분에서는 명세에 맞게 출력되도록 output을 맞춰줬다. 이 중 int와 int[]. void와 void[]를 구분하기 위해 TypeK 노드를 추가했으며 flag를 설정하여 일반 변수와 배열을 구분하도록 했다.

```c
/* procedure printTree prints a syntax tree to the
 * listing file using indentation to indicate subtrees
 */
static int flag = 0;
void printTree( TreeNode * tree )
{ int i;
  INDENT;
  while (tree != NULL) {
    if (tree->nodekind!=DeclK || tree->kind.decl!=TypeK) {
      printSpaces();
    }
    if (tree->nodekind==DeclK)
    { switch (tree->kind.decl) {
        case VarK:
          flag = 0;
          fprintf(listing,"Variable Declaration: name = %s, type = ",tree->attr.name);
          break;
        case ArrVarK:
          flag = 1;
          fprintf(listing,"Variable Declaration: name = %s, type = ",tree->attr.name);
          break;
        case FunK:
          fprintf(listing,"Function Declaration: name = %s, return type = ",tree->attr.name);
          break;
        case ParamK:
          flag = 0;
          if (strcmp(tree->attr.name,"none")==0) {
            fprintf(listing,"Void Parameter\n");
          }
          else {
            fprintf(listing,"Parameter: name = %s, type = ",tree->attr.name);
          }
          break;
        case ArrParamK:
          flag = 1;
          if (strcmp(tree->attr.name,"none")==0) {
            fprintf(listing,"Void Parameter\n");
          }
          else {
            fprintf(listing,"Parameter: name = %s, type = ",tree->attr.name);
          }
          break;
        case TypeK:
          if (flag == 0) {
            fprintf(listing,"%s\n",tree->attr.type);
          }
          else if (flag == 1) {
            fprintf(listing,"%s[]\n",tree->attr.type);
          }
          break;
        default:
          fprintf(listing,"Unknown DeclNode kind\n");
          break;
    }
  }
```

- cminus.y

주어진 명세에 맞게 cminus token의 흐름을 구현하는 부분으로 우선순위 및 conflict에 주의하여 작성하였다.

- type_specification

해당 부분은 변수의 type을 저장하기 위한 부분으로 구조체의 attr 속성에 type을 추가하여 해당 위치에 저장했다.

```
type_spec   : INT {
                $$ = newDeclNode(TypeK);
                $$->attr.type = "int";
            }
            | VOID {
                $$ = newDeclNode(TypeK);
                $$->attr.type = "void";
            }
            ;
```

- var_declaration

variable이 일반 변수인지 배열 변수인지 확인하기 위해 구분했으며 이에 따라 다른 노드를 생성하여 차이를 두었다. 이후 util.c에서 flag를 이용하여 배열에 대한 처리를 실시했다.

```
var_decl    : type_spec identifier SEMI {
                $$ = newDeclNode(VarK);
                $$->child[0] = $1;
                $$->attr.name = savedName;
            }
            | type_spec identifier LBRACE number RBRACE SEMI
            {
                $$ = newDeclNode(ArrVarK);
                $$->attr.name = savedName;
                $$->child[0] = $1;
            }
            ;
```

- param

variable과 마찬가지로 일반 변수인지 배열 변수인지 확인하기 위해 구분했으며 이에 따라 다른 노드를 생성하여 차이를 두었다.

```
param       : type_spec identifier {
                $$ = newDeclNode(ParamK);
                $$->child[0] = $1;
                $$->attr.name = savedName;
            }
            | type_spec identifier LBRACE RBRACE {
                $$ = newDeclNode(ArrParamK);
                $$->child[0] = $1;
                $$->attr.name = savedName;
            }
            ;
```

- shift-reduce conflict

cminus.y의 grammar를 과제 명세에 맞게 구현했지만 make를 통해 컴파일을 실시하자 shift-reduce conflict가 발생했다. 해당 conflict는 If와 If-Else로 인해 발생하는 것이다. input으로 if를 읽고난 후 reduce를 실시할지 else로 shift를 실시할지에 대해 문법으로 정의하지 않았기 때문이다. 해당 부분에 대해 찾아본 결과 %nonassoc을 이용하여 conflict를 없앨 수 있다고 했다. 이에 따라 해당 부분을 추가했다.

```
%nonassoc LOWER_THAN_ELSE
%nonassoc ELSE
```

```
select_stmt : IF LPAREN expression RPAREN statement %prec LOWER_THAN_ELSE {
                $$ = newStmtNode(IfK);
                $$->child[0] = $3;
                $$->child[1] = $5;
                $$->child[2] = NULL;
            }
            | IF LPAREN expression RPAREN statement ELSE statement {
                $$ = newStmtNode(IfK);
                $$->child[0] = $3;
                $$->child[1] = $5;
                $$->child[2] = $7;
            }
            ;
```

3. Sample screenshot

- test.1.txt

```
/mnt/c/Users/pyo99/CS/Compiler/2021_ele4029_2019094511/2_Parser | master ?40  ./cminus_parser test.1.txt

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
```

- test.2.txt

```
/mnt/c/Users/pyo99/CS/Compiler/2021_ele4029_2019094511/2_Parser | master ?40  ./cminus_parser test.2.txt

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
```

- test.3.txt

```
/mnt/c/Users/pyo99/CS/Compiler/2021_ele4029_2019094511/2_Parser | master ?40  ./cminus_parser test.3.txt

TINY COMPILATION: test.3.txt

Syntax tree:
  Function Declaration: name = main, return type = int
    Parameter: name = a, type = void[]
    Compound Statement:
      Variable Declaration: name = b, type = void
      Variable Declaration: name = c, type = int
      Assign:
        Variable: name = d
          Const: 1
        Op: +
          Variable: name = b
          Variable: name = c
```

해당 코드는 project resource가 아닌 과제 명세에 있는 코드로 semantic과 무관하게 parsing이 잘 일어나는 것을 확인했다.