Compiler assignment1

학과 : 컴퓨터소프트웨어학부

학번 : 2017029516

이름 : 김태환

- **Compilation environment and method**

  Oracle VM VirtualBox, ubuntu 18.04.6 에서 작업, gcc를 이용하여 컴파일 하였습니다.

  Bison version은 3.0.4입니다. Gcc version은 7.5.0입니다.

- **Brief explanations about how to implement and how it operates**

  명세에 맞추어 총 4가지 파일을 수정하였습니다. main.c, global.h, util.c, cminus.y

  먼저 main.c에서는 syntax tree만을 print하도록

  #define NO_PARSE FALSE

  #define NO_ANALYZE TRUE

  Int TraceParse = TRUE;

  의 수정을 하였습니다.

- 그다음으로, global.h에서는

```
typedef enum {StmtK,ExpK} NodeKind;
typedef enum {CompK,IfK,ElseK,WhileK,ReturnK} StmtKind;
typedef enum {VarDecK,FunDecK,AssignK,CallK,OpK,ConstK,VarK,ParamK} ExpKind;

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

  Appendix를 참고하여, 종류를 구분하여 출력해야하는 모든 case들을 StmtKind와 ExpKind로 추가했습니다. 이때 if 와 if else는 구분하여 IfK와 ElseK로 나누었습니다.

  또한, 명세에따라 타입이 총 4가지(void,int,void[],int[])이므로 expType에 추가해주었습니다.

또, 배열타입이 추가되었기 때문에 배열의 size를 가지는 요소를 treeNode 구조체에 넣어 주었습니다.

- 그다음으로 util.c에서는

```
char *tokenToType[] = {"void","int","void[]","int[]"};

void printTree( TreeNode * tree )
{ int i;
  INDENT;
  while (tree != NULL && !Error) {
    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 ElseK:
                    fprintf(listing,"If-Else Statement:\n");
                    break;
                case WhileK:
                    fprintf(listing,"While Statement:\n");
                    break;
                case ReturnK:
                    if(tree->child[0] == NULL)
                        fprintf(listing,"Non-value Return Statement:\n");
                    else
                    fprintf(listing,"Return Statement:\n");
                    break;
                default:
                    fprintf(listing,"Unknown ExpNode kind\n");
                    break;
            }
    }
```

먼저 Void,Integer등의 값을 출력예시에 맞는 타입으로 변환해주기위해 tokenToType을 선언하여 알맞은 타입으로 매칭되도록 하였습니다.

그리고 printTree의 while문의 조건에 !Error을 추가하여 에러가 발생시에는 tree를 출력하지 않도록 구현하였습니다.

이외 구현은 Kind에 따라 명세에 나온대로 문장을 출력하였습니다.

```
  else if (tree->nodekind==ExpK)
  { switch (tree->kind.exp)
          {
              case VarDecK:
                  fprintf(listing,"Variable Declaration: name = %s, type = %s\n", tree->attr.name, tokenToType[tree->type]);
                  break;
              case FunDecK:
                  fprintf(listing, "Function Declaration: name = %s, return type = %s\n", tree->attr.name, tokenToType[tree->type]);
                  break;
              case CallK:
                  fprintf(listing,"Call: function name = %s\n",tree->attr.name);
                  break;
              case OpK:
                  fprintf(listing,"Op: ");
                  printToken(tree->attr.op,"\0");
                  break;
              case AssignK:
                  fprintf(listing,"Assign:\n");
                  break;
              case ConstK:
                  fprintf(listing,"Const: %d\n",tree->attr.val);
                  break;
              case VarK:
              fprintf(listing,"Variable: name = %s\n",tree->attr.name);
                  break;
              case ParamK:
                  if(tree->type != NULL)
                      fprintf(listing, "Parameter: name = %s, type = %s\n", tree->attr.name, tokenToType[tree->type]);
                  else
                      fprintf(listing, "Void Parameter\n");
                  break;
      default:
        fprintf(listing,"Unknown ExpNode kind\n");
        break;
    }
  }
```

- 그다음으로, cminus.y에서는 definition과 rule section을 명세에 있는 BNF Grammar에 맞추어 수정해주었습니다.

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

Definition 섹션에서 추가로 OTHER을 추가하였습니다. 이는 후에 dangling else를 설명하며 나옵니다..

```
id                      : ID
                        { $$ = newExpNode(VarK);
                          $$->attr.name = copyString(tokenString);
                        }
                        ;

num                     : NUM
                        { $$ = newExpNode(ConstK);
                          $$->attr.val = atoi(tokenString);
                        }
                        ;
```

또 Appendix에는 따로 명시되어있지 않지만 id와 num을 따로 만들어주어 각각variable과 Const로 노드를 생성해주는 방식으로 구현했습니다.

```
selection_stmt          : IF LPAREN expression RPAREN stmt %prec OTHER
                        { $$ = newStmtNode(IfK);
                          $$->child[0] = $3;
                          $$->child[1] = $5;
                        }
                        | IF LPAREN expression RPAREN stmt ELSE stmt
                        {   $$ = newStmtNode(ElseK);
                            $$->child[0] = $3;
                            $$->child[1] = $5;
                            $$->child[2] = $7;
                        }
                        ;
```

Ambiguity를 해결하기 위해서 Yacc관련 문서를 구글링하던중

```
%token NAME NUMBER
%left '-' '+'
%left '+' '/'
%nonassoc UMINUS
expression:     expression '+' expression
        |       expression '-' expression
        |       expression '+' expression
        |       expression '/' expression
        |       '-' expression %prec UMINUS
        |       '(' expression ')'
        |       NUMBER
        ;
```

위에서 중간에 보이는 %prec UMINUS는 '-' expression과 같은 문자열을 파싱하게 되면 -에 대해서는 UMINUS로 지정된 우선순위를 사용하라는 의미이다. 따라서, 모호한 문법에서 쉽게 해결할 수가 있게 된다.

http://nlp.kookmin.ac.kr/sskang/lect/compiler/lexyacc/Lex_Yacc.htm

를 참고하여 %prec을 이용하여 OTHER와 IF문의 precedence를 OTHER로 지정된 우선순위를 사용하게 하여 IF-ELSE문이 더 높은 우선순위를 가지는 방식으로 해결하였습니다.

-Test Case

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

```
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
```

Dangling else problem

```
C-MINUS COMPILATION: dangle.txt

Syntax tree:
  Function Declaration: name = main, return type = void
    Void Parameter
    Compound Statement:
      If Statement:
        Op: <
          Variable: name = a
          Const: 0
        If-Else Statement:
          Op: >
            Variable: name = a
            Const: 3
          Assign:
            Variable: name = a
            Const: 3
          Assign:
            Variable: name = a
            Const: 4
```