Project #2. Parser Report

2018009234

1. 컴파일 환경

Windows Subsystem for Linux를 이용하여 Ubuntu 20.04.5 LTS에서 작성된 프로젝트입니다. 과제 명세서에 작성되어 있는 Makefile를 이용하기 때문에 gcc를 이용하여 컴파일 합니다. make all 명령어로 cminus_cimpl과 cminus_lex 두 개의 실행 파일을 만들 수 있습니다. make cminus_parser 명령어로 cminus_parser 실행 파일을 만들 수 있습니다.

2. 과제 분석

Yacc를 이용하여 C-Minus Grammar를 가지는 Parser를 생성하는 것이 이번 과제의 목표입니다. 과제 명세에 맞게끔, globals.h, util.c 등을 수정하고 Yacc의 input인 cminus.y를 작성했습니다.

Report에서는 전체적인 구현, Attribute, Dangling Else Problem, Error Handling를 중점으로 작성하였습니다. 몇 번 Rule인지는 cminus.y에 주석으로 달았습니다.

3. 전체적인 구현

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

C-Minus의 AST에 필요한 노드를 위해 enum을 수정하였습니다. 수정한 enum과 과제 명세의 출력 양식에 맞게끔 printTree()를 수정하였습니다. DeclKind를 가지는 노드들을 생성하는 함수인 newDeclNode()를 정의하였습니다.

cminus.y을 작성할 때에는 . ID와 NUM에 대한 Rule30, 31을 추가한 것을 제외한다면 과제 명세서 3페이지에 적힌 BNF for C-Minus를 최대한 변형하지 않고 작성하였습니다. 각 Rule에 대한 Action은 노드를 생성해야할 때에는 생성하거나 $1, $2등을 통해 기존 노드들에 접근하는 등 기존 tiny.y를 참고하여 작성하였습니다.

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

Ambiguity를 해결하기 위해서 연산자들의 우선 순위를 위와같이 지정하였습니다.

4. Attribute

Function declaration에서의 type, 연산자들, NUM은 Rule5, Rule21, Rule23, Rule25, Rule31에서 노드를 생성하면서 Attribute를 저장하기 때문에 쉽게 해결할 수 있었습니다. Variable access, function call에서의 ID는 문제가 되지 않지만 function declaration에서는 tiny.y에서처럼 Mid-Rule Action을 이용한다면, compound statement에서 parameters, variable declaration이 수행될 수 있기 때문에 문제가 생길 수 있습니다. 따라서 Rule30을 추가하여 ID가 나올 때마다 노드를 생성하고 Variable access, function call에서는 생성한 노드를 그대로 사용하고, function/variable declaration, parameter처럼 ID의 string만을 필요한 경우에는 string을 복사한 후 free해주는 방식을 이용하였습니다.

```
// Rule6
func_decl   : type_spec identifier LPAREN params RPAREN comp_stmt
                {
                    $$ = $1;
                    $$->attr.name = copyString($2->attr.name);
                    $$->lineno = $2->lineno;
                    free($2);
                    $$->kind.decl = FuncK;
                    $$->child[0] = $4;
                    $$->child[1] = $6;
                }
            ;
```

5. Dangling Else Problem

```
// Rule15
selec_stmt  : IF LPAREN exp RPAREN stmt %prec THEN
                {
                    $$ = newStmtNode(IfK);
                    $$->child[0] = $3;
                    $$->child[1] = $5;
                }

            | IF LPAREN exp RPAREN stmt ELSE stmt
                {
                    $$ = newStmtNode(IfElseK);
                    $$->child[0] = $3;
                    $$->child[1] = $5;
                    $$->child[2] = $7;
                }
            ;
```

Dangling Else Problem을 해결하기 위해서 ELSE가 있는 Production에 우선 순위를 줘야 합니다. 따라서 ELSE가 없는 Production을 %prec을 이용하여 THEN과 같은 우선 순위로 취급하였고, THEN은 ELSE보다 낮은 우선 순위로 설정하였습니다.

6. Error Handling

기본적으로 C-Minus BNF가 Accept하지 않는 문법의 경우(ERROR 토큰이 있다거나, 문법적으로 틀린 경우) Start symbol, 즉 program이 될 수 없기 때문에 savedTree에 노드가 저장되지 않아 Syntax tree가 출력되지 않습니다. 하지만 int main(void) { }}의 경우 int main(void){ }가 program이 된 후 }에서 에러가 발생하는 것으로 분석하였기 때문에, 에러도 존재하고 Syntax tree가 출력되는 문제가 있었습니다.

```
C-MINUS COMPILATION: testcase.txt
Syntax error at line 1: syntax error
Current token: }

Syntax tree:
  Function Declaration: name = main, return type = int
    Void Parameter
    Compound Statement:
```

따라서 아래와 같이 현재 Error가 없는 경우에만 savedTree에 만들어진 노드를 저장하고, error가 발생했다면 savedTree를 NULL로 바꾸는 방식으로 Rule1을 수정하였습니다.

```
// Rule1
program     : decl_list
              {
                  if(!Error) {
                      savedTree = $1;
                  }
              }
            | error
              {
                  savedTree = NULL;
              }
            ;
```

7. 테스트 케이스 결과

```
int globalVar1;

int testFunc1 (void){
    int localVar1;
    int localVar2;
    int localVar3[2];

    localVar1 = localVar1 + localVar3[0];
    localVar3[1] = localVar1 / localVar2 - localVar1*localVar3[1];

    if(localVar1 <= localVar2)
        localVar1 = localVar2 = localVar1 + (localVar2 - localVar1) * localVar3[1];

    if(localVar1 < localVar2)
        localVar1 = localVar2;

    if(localVar1 > localVar2)
        localVar1 = localVar2;

    if(localVar1 >= localVar2)
        localVar1 = localVar2;

    if(localVar1 == localVar2)
        localVar1;
    else {
        int localVar4;
        int localVar5[3];
        testFunc2(localVar1+localVar4, localVar3, localVar4, localVar5);
    }

    while (localVar1 != localVar2) { ; }

    return localVar3[localVar1];
}

int globalVar2[3];

void testFunc2 (int param1, int param2[], void param3, void param4[]) {
    return;
}

int main(void) {
    testFunc1();
}
```

```
C-MINUS COMPILATION: testcase.txt

Syntax tree:
  Variable Declaration: name = globalVar1, type = int
  Function Declaration: name = testFunc1, return type = int
    Void Parameter
    Compound Statement:
      Variable Declaration: name = localVar1, type = int
      Variable Declaration: name = localVar2, type = int
      Variable Declaration: name = localVar3, type = int[]
        Const: 2
      Assign:
        Variable: name = localVar1
        Op: +
          Variable: name = localVar1
          Variable: name = localVar3
            Const: 0
      Assign:
        Variable: name = localVar3
          Const: 1
        Op: -
          Op: /
            Variable: name = localVar1
            Variable: name = localVar2
          Op: *
            Variable: name = localVar1
            Variable: name = localVar3
              Const: 1
      If Statement:
        Op: <=
          Variable: name = localVar1
          Variable: name = localVar2
        Assign:
          Variable: name = localVar1
          Assign:
            Variable: name = localVar2
            Op: +
              Variable: name = localVar1
              Op: *
                Op: -
                  Variable: name = localVar2
                  Variable: name = localVar1
                Variable: name = localVar3
                  Const: 1
      If Statement:
        Op: <
          Variable: name = localVar1
          Variable: name = localVar2
        Assign:
          Variable: name = localVar1
          Variable: name = localVar2
      If Statement:
        Op: >
          Variable: name = localVar1
          Variable: name = localVar2
        Assign:
          Variable: name = localVar1
          Variable: name = localVar2
      If Statement:
        Op: >=
          Variable: name = localVar1
          Variable: name = localVar2
        Assign:
          Variable: name = localVar1
          Variable: name = localVar2
      If-Else Statement:
        Op: ==
          Variable: name = localVar1
          Variable: name = localVar2
        Variable: name = localVar1
        Compound Statement:
          Variable Declaration: name = localVar4, type = int
          Variable Declaration: name = localVar5, type = int[]
            Const: 3
          Call: function name = testFunc2
            Op: +
              Variable: name = localVar1
              Variable: name = localVar4
            Variable: name = localVar3
            Variable: name = localVar4
            Variable: name = localVar5
      While Statement:
        Op: !=
          Variable: name = localVar1
          Variable: name = localVar2
        Compound Statement:
      Return Statement:
        Variable: name = localVar3
          Op: +
            Variable: name = localVar1
            Const: 1
  Variable Declaration: name = globalVar2, type = int[]
    Const: 3
  Function Declaration: name = testFunc2, return type = void
    Parameter: name = param1, type = int
    Parameter: name = param2, type = int[]
    Parameter: name = param3, type = void
    Parameter: name = param4, type = void[]
    Compound Statement:
      Non-value Return Statement
  Function Declaration: name = main, return type = int
    Void Parameter
    Compound Statement:
      Call: function name = testFunc1
```

다양한 경우를 담은 테스트 케이스를 제작하여 결과를 출력한 결과, 정상적으로 Parser가 동작하는 것으로 분석하였습니다.