ELE4029 Project Report

2. Parser

박준영

1. How to build

1.1 Compilation Environment

본 프로젝트는 다음의 환경에서 개발, 테스트하였다.

- Ubuntu Server 20.04.3 LTS
- gcc version 9.3.0
- GNU Make 4.2.1
- flex 2.6.4
- GNU bison 3.5.1

1.2 Compilation Method

우선 아래 명령어를 통해 hconnect에서 repository를 clone한다.

```
git clone https://hconnect.hanyang.ac.kr/2021_ele4029_11216/2021_ele4029_2019064811
```

이후 다음 명령어들을 수행하여 컴파일한다.

```
cd 2021_ele4029_2019064811/2_Parser
make
```

컴파일이 완료되면 cminus_parser 실행파일이 생성된다. 프로그램의 실행은 다음과 같은 방식으로 한다.

```
./cminus_parser <filename>
```

2. Implementation

1) Node Kind

Statement Node와 Expression Node에 대하여 각각 아래와 같이 kind enumerator를 만들었다. 이때 statement는 어떤 값을 내보내지 않는 노드, expression은 어떤 값을 산출할 수 있는 노드로 구분 지었다.

- StmtKind
  - VoidParamK : void parameter statement
  - ParamK : non-void parameter statement
  - VarDeclK : variable declaration statement
  - FuncDeclK : function declaration statement
  - CompundK : compound statement
  - ReturnK : return statement
  - WhileK : while statement
  - IfK : if statement
  - IfElseK : if-else statement
- ExpKind
  - OpK : operation expression
  - ConstK : const value expression
  - AssignK : assignment expression
  - VarAccessK : variable access expression
  - CallK : function call expression

2) Expression Type

Expression의 값의 종류를 다음과 같이 정의하였다.

- Void : void type (void) / VoidArr : void array type (void[])
- Integer : integer type (int) / IntArr : integer array type (int[])
- Boolean : boolean type (bool)

3) Scanner Modification

yacc로 만들어진 파서는 LALR(1)로, 상황에 따라 lookahead token을 추가로 본다. 그러나 기존의 scanner는 getToken이 호출되면 무조건 currentToken에 overwrite하기 때문에 lookahead를 볼 경우 원래의 currentToken이 손상되는 문제점이 존재한다. 따라서 currentToken을 두 개 두어 getToken이 호출될 때마다 두 currentToken을 switch하여 쓰도록 하여 lookahead를 보더라도 기존의 currentToken이 유실되지 않도록 하였다.

4) Parser

parser는 과제명세 Appendix A.2의 BNF Grammar를 참고하여 구현하였다. 다만 기존의 parser 코드에서 saved* 변수의 경우, 여러 state로 전이되는 과정에서 save를 여러 번 해야 하는 특정 문법을 처리할 때 overwrite되는 문제가 존재한다. 여러 state로 전이되더라도 마지막으로 전이된 state부터 역으로 처리되는 점에 착안하여 saved* 변수를 모두 stack 자료구조로 구현하였다.

3. Examples

```
/* A program to perform Euclid's
    Algorithm to computer gcd */

int gcd (int u, int v)
{
        if (v == 0) return u;
        else return gcd(v, u-u/v*v);
        /* u-u/v*v == u mod v */
}

void main(void)
{
        int x; int y;
        x = input(); y = input();
        output(gcd(x,y));
}
```

```
(base) jun@deepjun:~/Univ/ELE4029/2_Parser$ ./cminus_parser testcase/test.1.txt

C-MINUS COMPILATION: testcase/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
```

```
void main(void)
{
        int i; int x[5];

        i = 0;
        while( i < 5 )
        {
                x[i] = input();

                i = i + 1;
        }

        i = 0;
        while( i <= 4 )
        {
                if( x[i] != 0 )
                {
                        output(x[i]);
                }
        }
}
```

```
(base) jun@deepjun:~/Univ/ELE4029/2_Parser$ ./cminus_parser testcase/test.2.txt

C-MINUS COMPILATION: testcase/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
```

```
void main(void)
{
    if (a < 0)
    if (a > 3) a = 3; else a = 4;
}
```

```
(base) jun@deepjun:~/Univ/ELE4029/2_Parser$ ./cminus_parser testcase/test.0.txt

C-MINUS COMPILATION: testcase/test.0.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
```