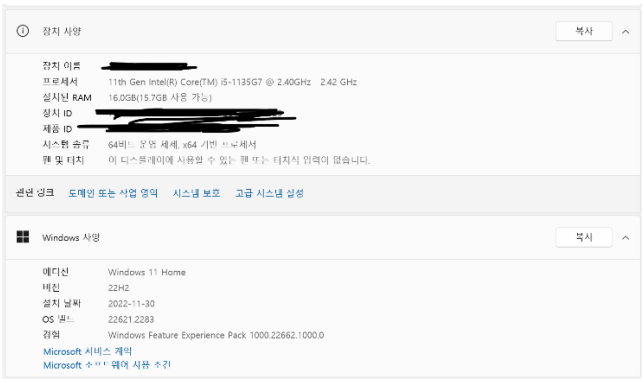
컴파일러설계 2_Parser

2019082279 김영현

1. Compilation environment and method



윈도우에서 WSL(리눅스용 윈도우 하위 시스템) 사용, vscode로 작업, ubuntu 20.04

```
kyh011@DESKTOP-944FOPD:/mnt/c/Users/KYH/StudentID-2/StudentID$ ls
1_Scanner  execute..txt
kyh011@DESKTOP-944FOPD:/mnt/c/Users/KYH/StudentID-2/StudentID$ lsb_release -a
No LSB modules are available.
Distributor ID: Ubuntu
Description:    Ubuntu 20.04.6 LTS
Release:        20.04
Codename:       focal
```

make clean -> make all -> ./cminus_parser ./example/test.1.txt

2. Brief explanations about how to implement and how it operates

cimus.y를 수정했다. Cimus.y에 기존에 작성되어 있었던 코드와 pdf의 다이어그램을 참고했다. Globals.h와 util.c를 참고했다.

```
fun_declaration     : type_specifier identifier LPAREN params RPAREN compound_stmt
```

fun_declaration은 기존에 작성되어 있던 val_declaration의 구조를 참고했다. Pdf를 보면 val_declaration과 fun과의 차이점은 child의 유무였고, 기존의 val_declaration의 code에 child 2개를 설정해주는 것을 추가했다.

```
param_list          : param_list COMMA param
```

params는 param_list와 param이 핵심이었다. 특히 param_list의 경우 left-recursive 구조이며, 이와 비슷한 구조인 declaration_list의 구조를 참고했다. Declaration_list와의 차이점은 중간에 COMMA가 들어가서 sibling이 $1, $3이라는 것이었다.

```
param               : type_specifier identifier
```

param 또한 type과 name만 변경해주면 되었다. Param의 경우 val_declaration의 구조를 참고했다. Child만 추가하지 않았다.

```
params              : param_list { $$ = $1; }
                    | VOID
```

Params의 void parameter에서 많이 헤맸다. 처음엔 params 노드를 만들고 type을 void로 설정해줬다. 하지만 출력될 경우 기존의 일반적인 params와 동일하게 출력됐다. Util.c를 보고 flag의 의미를 찾아서 params 노드를 만들고 flag를 1로 설정해주니 void Parameter로 출력됐다.

```
compound_stmt       : LCURLY local_declarations statement_list RCURLY
```

compound_stmt는 pdf의 다이어그램 구조를 보고 node를 만들고 child만 설정해줬다.

```
local_declarations  : local_declarations var_declaration
```

local_declarations또한 val_declaration의 구조를 참고했다.

```
statement_list      : statement_list statement
```

statement_list 또한 val_declaration의 구조를 참고했다.

```
selection_stmt      : IF LPAREN expression RPAREN statement ELSE statement
```

selection_stmt의 경우 pdf 다이어그램의 경우에 따라 child 개수만 달리했다. 거기에 flag를 바꿨다.

```
iteration_stmt      : WHILE LPAREN expression RPAREN statement
```

iteration_stmt의 경우 node를 만들고 child만 할당했다.

```
return_stmt         : RETURN SEMI
```

return_stmt도 node를 만들고 child만 배치했다.

```
expression          : var ASSIGN expression
```

expression도 node를 만들고 child만 배치했다.

```
var                 : identifier
```

var의 경우 node를 만들고 name을 설정해주고 child를 설정했다.

```
simple_expression   : additive_expression relop additive_expression
```

simple_expression의 경우 pdf를 보고 opcode를 설정하고 child를 배정했다.

```
additive_expression : additive_expression addop term
```

additive_expression또한 동일하게 했다.

```
addop
```

addop와 relop는 relop의 이미 작성된 코드를 참고해서 작성했다.

```
term                : term mulop factor
```

term또한 simple_expression과 additive_expression과 동일하게 작성했다.

```
mulop
```

addop와 relop의 구조를 참고했다.

```
call                : identifier LPAREN args RPAREN
```

pdf의 다이어그램을 보고 name을 설정하고 child를 설정했다.

전체적으로 알맞은 NodeKind 의 노드를 설정하고 해당 TreeNode의 name이나 type 등을 설정해주는 방법은 간단했다. 하지만 left-recursive구조의 경우 기존에 작성되어 있던 코드를 참고하지 않았다면 꽤나 오래 헤맸을 것 같다.

3. Examples and corresponding result screenshots

test1과 test2 모두 result파일과 동일한 결과를 보여줬다.

```
kyh011@DESKTOP-944FOPD:/mnt/c/Users/KYH/2019082279/student_id/2_Parser$ ./cminus_parser ./example/test.1.txt

C-MINUS COMPILATION: ./example/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
```

```
kyh011@DESKTOP-944FOPD:/mnt/c/Users/KYH/2019082279/student_id/2_Parser$ ./cminu
./example/test.2.txt

C-MINUS COMPILATION: ./example/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
```