Compiler Project #2. Parser Report

2018008277 노주찬

1. Compilation Environment

아래의 환경에서 프로젝트를 컴파일하고 테스트했습니다.

항목	값
OS	Ubuntu 20.04.4 LTS
Architecture	x86_64
C Compiler	gcc (Ubuntu 9.4.0-1ubuntu1~20.04.1) 9.4.0
Lex/Flex	flex 2.6.4
Yacc/Bison	Bison 3.8.1 (http://ftp.gnu.org/gnu/bison/)

apt에서 기본으로 설치되는 bison 대신 더 최신 버전의 bison을 설치해서 사용하였습니다. 또한 bison에서 제공하는 추가적인 기능을 사용하기 위해 POSIX yacc를 emulate 하지 않도록 Makefile 중 y.tab.c 빌드 커맨드의 yacc을 bison으로 수정했습니다.

2. Scanner Modification

AST에 정보를 저장하기 위해 number 토큰이 나타내는 integer 값과 ID 토큰이 나타내는 identifier string 정보가 필요했습니다. 기존 Tiny 구현에서는 mid rule과 전역으로 정의된 savedName을 사용해 identifier string을 저장하고 있었지만, mid rule을 사용하지 않기 위해 scanner 단계에서 number, ID 토큰에 syntax value를 지정하도록 수정했습니다.

```
56  56    "}"             {return RCURLY;}
57  57    ";"             {return SEMI;}
58  58    ","             {return COMMA;}
59      - {number}        {return NUM;}
60      - {identifier}    {return ID;}
    59  + {number}        {yylval.num_value = atoi(yytext); return NUM;}
    60  + {identifier}    {yylval.id_name = copyString(yytext); return ID;}
61  61    {newline}       {lineno++;}
62  62    {whitespace}    {/* skip whitespace */}
63  63    .               {return ERROR;}
```

3. Bison

3.1. %union

앞서 언급한 ID, NUM 토큰 이외에도 구현의 편의를 위해서 TreeNode* 이외의 다른 타입의 syntax value를 사용할 필요가 있었기 때문에, 아래와 같이 %union을 정의하여 syntax value를 사용했습니다.

```
%union {
 TreeNode *node;
 char* id_name;
 ExpType type_spec;
 int num_value;
 TokenType op_type;
}
```

이름	설명
TreeNode* node	아래 기술된 특수한 경우를 제외한 기본 Syntax Value 타입입니다.
char* id_name	ID 토큰인 경우, identifier string을 저장합니다.
ExpType type_spec	Array var_decl을 용이하게 처리하기 위해 type_spec 토큰에 type 정보(int 또는 void)를 저장합니다.
int num_value	NUMBER 토큰인 경우, integer 값을 저장합니다.
TokenType op_type	relop, addop, mulop 등의 토큰에 binary operator를 저장합니다.

3.2. Dangling-Else (Closest-If)

Dangling Else 문제를 해결하기 위해서 2_Syntax_Analysis_1_1의 마지막 슬라이드를 참고하여 Statement를 if-else 사이의 statement 위치로 들어갔을 때 뒤에서 등장하는 else와 매칭이 가능한 if statement (unmatched statement)가 포함된 statement와 그렇지 않은 statement로 분리한 다음, unmatched statement가 if와 else 사이의 statement 위치에 등장하지 않도록 문법을 수정했습니다.

4. globals.h (ast.h)

%union에서 사용되는 타입 정의가 y.tab.h를 include 하기 전에 이뤄져야 했기 때문에, TreeNode에 관련된 부분을 ast.h라는 별도의 헤더 파일로 분리하여 globals.h에서 include 하도록 수정했습니다.

ast.h에서는 대부분 주어진 문법에 맞게 Node와 Kind를 작성하고, 필요한 정보를 저장할 수 있도록 TreeNode* 내의 멤버를 수정했습니다. 예외적으로 기존 Tiny 구현에서 같은 종류의 node (Declaration, Parameter, Statement 등) 가 반복되는 list를 저장할 때 매번 sibling을 계속해서 타고 들어가서 저장하는 부분이 비효율적이라고 생각되었습니다. 이 부분을 개선하기 위해 Linked List 형태로 TreeNode를 저장하는 ListNode를 추가했습니다. ListNode에서는 첫번째 child를 list의 첫번째 node로, 두번째 child를 list

의 마지막 node로 두는 식으로 구현하여, AST 출력과정에서 ListNode는 Indent 처리를 하지 않고, 두번째 child는 순회하지 않도록 처리했습니다.

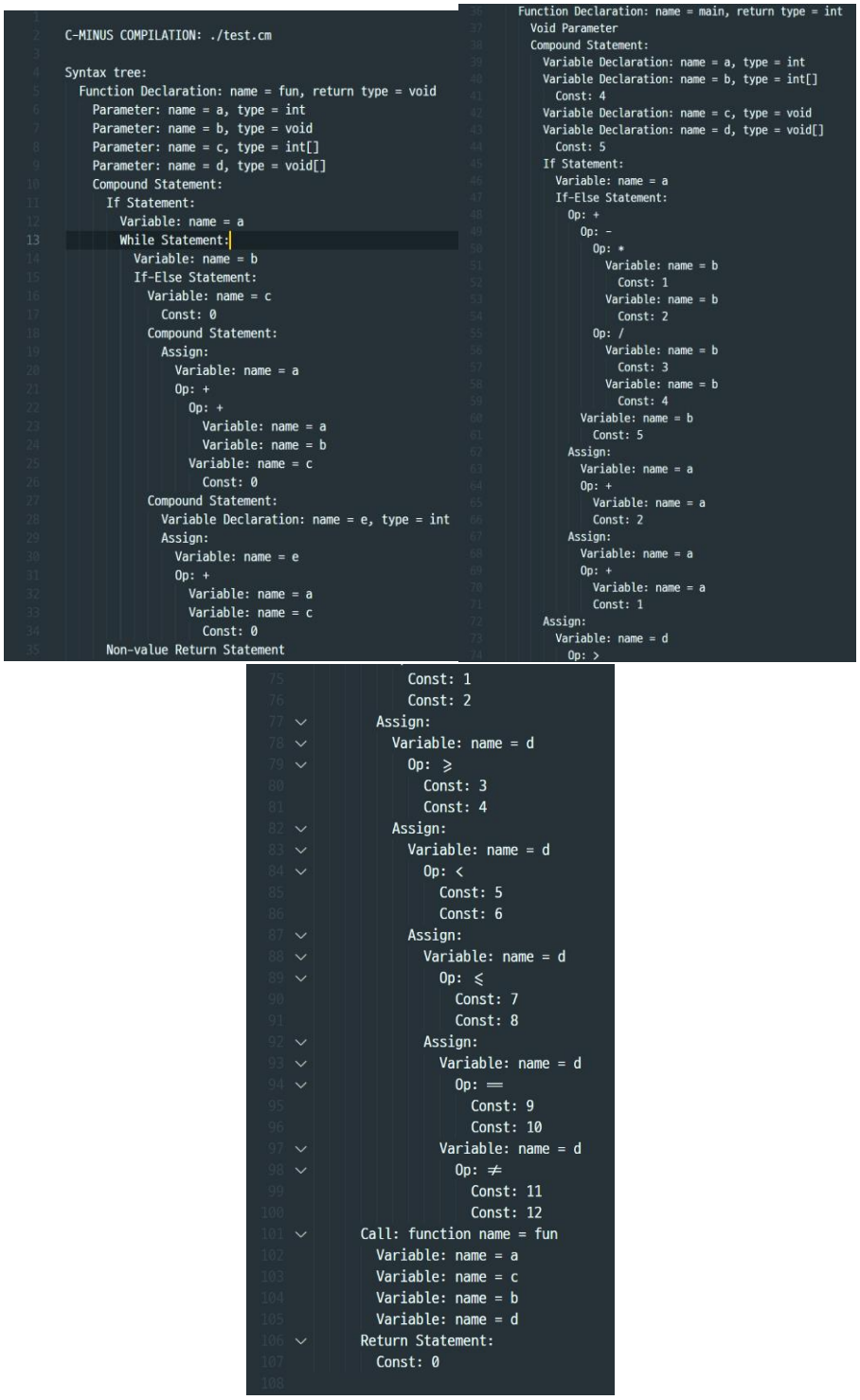
5. Examples and Results

```
1   int main ( void ) {
2       a ! b; /* unrecognized op '!' */
3   }
4
```

```
❯ ./cminus_parser test.cm

C-MINUS COMPILATION: test.cm
Syntax error at line 2: syntax error
Current token: ERROR: !

Syntax tree:
```

먼저 ERROR 토큰이 잘 처리되는지 확인하기 위해 C-Minus에 정의되지 않은 logical NOT operator이 등장하는 Example로 테스트를 해보았고, 정상적으로 ERROR 토큰이 발견되는 즉시 Syntax error가 발생하는 것을 확인했습니다.

```
/* fun decl test */
void fun (int a, void b, int c[], void d[]) {
    /* dangling-else with unmatched if statement in a while statement */
    if (a) while (b) if (c[0]) {
        /* associativity test */
        a = a + b + c[0];
    } else
    /* cmpnd statement test */
    {
        int e;
        e = a + c[0];
    }

    /* non-value return test */
     return ;
}

/* void-parameter test */
int main (void) {
    /* local_decles test */
    int a;
    int b[4];
    void c;
    void d[5];

    /* if, if-else, addop, mulop, dangling else test */
    if (a) if (b[1] * b[2] - b[3] / b[4] + b[5]) a = a + 2; else a = a + 1;

    /* var, relop test */
    d[1 > 2] =  d[3 ≥ 4] = d[5 < 6] = d[7 ≤ 8] = d[9 == 10] = d[11 ≠ 12];

    /* function call test */
    fun(a, c, b, d);

    /* return test */
    return 0;
}
```



```
C-MINUS COMPILATION: ./test.cm

Syntax tree:
  Function Declaration: name = fun, return type = void
    Parameter: name = a, type = int
    Parameter: name = b, type = void
    Parameter: name = c, type = int[]
    Parameter: name = d, type = void[]
    Compound Statement:
      If Statement:
        Variable: name = a
        While Statement:
          Variable: name = b
          If-Else Statement:
            Variable: name = c
              Const: 0
            Compound Statement:
              Assign:
                Variable: name = a
                Op: +
                  Op: +
                    Variable: name = a
                    Variable: name = b
                  Variable: name = c
                    Const: 0
            Compound Statement:
              Variable Declaration: name = e, type = int
              Assign:
                Variable: name = e
                Op: +
                  Variable: name = a
                  Variable: name = c
                    Const: 0
      Non-value Return Statement
Function Declaration: name = main, return type = int
  Void Parameter
  Compound Statement:
    Variable Declaration: name = a, type = int
    Variable Declaration: name = b, type = int[]
      Const: 4
    Variable Declaration: name = c, type = void
    Variable Declaration: name = d, type = void[]
      Const: 5
    If Statement:
      Variable: name = a
      If-Else Statement:
        Op: +
          Op: -
            Op: *
              Variable: name = b
                Const: 1
              Variable: name = b
                Const: 2
            Op: /
              Variable: name = b
                Const: 3
              Variable: name = b
                Const: 4
          Variable: name = b
            Const: 5
        Assign:
          Variable: name = a
          Op: +
            Variable: name = a
            Const: 2
        Assign:
          Variable: name = a
          Op: +
            Variable: name = a
            Const: 1
    Assign:
      Variable: name = d
        Op: >
          Const: 1
          Const: 2
      Assign:
        Variable: name = d
          Op: ≥
            Const: 3
            Const: 4
        Assign:
          Variable: name = d
            Op: <
              Const: 5
              Const: 6
          Assign:
            Variable: name = d
              Op: ≤
                Const: 7
                Const: 8
            Assign:
              Variable: name = d
                Op: ==
                  Const: 9
                  Const: 10
              Variable: name = d
                Op: ≠
                  Const: 11
                  Const: 12
    Call: function name = fun
      Variable: name = a
      Variable: name = c
      Variable: name = b
      Variable: name = d
    Return Statement:
      Const: 0
```

Dangling-else를 포함하여 여러 경우를 포함한 테스트 케이스에서도 의도된 대로 동작하는 것을 확인했습니다.