CS3300: Compiler Design **Odd Sem**

Assignment #2

Abstract Syntax Tree Deadline: 26/08/2018, 11:55PM

Task

The aim of the assignment is to create an Abstract Syntax Tree for the language used in Assignment #1 and perform queries on them.

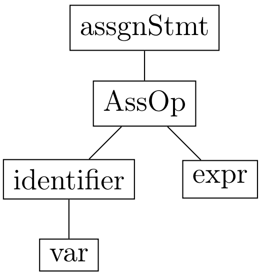
The Abstract Syntax Tree for various C constructs are as follows: (Note: expr<i> are instances of 'expr' numbered just for clarity.)

- Assignment Statement:

```
assgnStmt -> var AssOp expr
```

AssOp → { = }

AST:

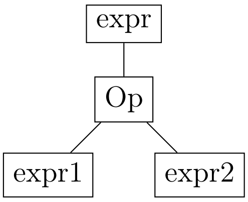


- Expressions:
  - Binary Operators

```
expr -> expr1 Op expr2
```

Op → { Relational operators, Binary Operators, Logical Operators }

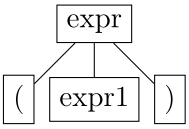
AST:



  - Parentheses expression

```
expr -> (expr1)
```

AST:

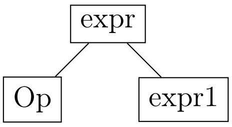


  - Unary Operators

```
expr -> Op expr1
```

Op → { Unary Operators }

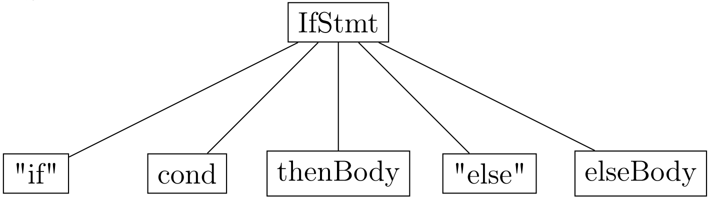
AST:



- If Statement:

```
    IfStmt -> if (cond){
                      thenBody
                  }else{
                      elseBody
                  }
```

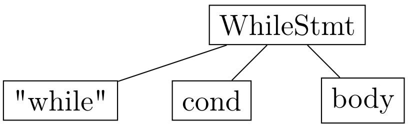
AST:



- While Statement:

```
    WhileStmt -> While (cond){
                      body
                   }
```

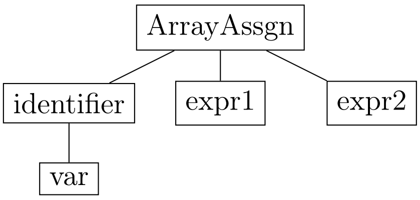
AST:



- Array Assignment Statement:

```
ArrayAssgn -> var [expr1] = expr2;
```

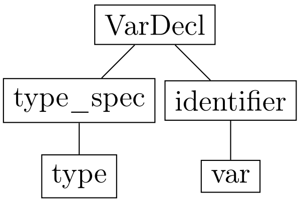
AST:



- Variable Declaration:

```
VarDecl -> type var;
```

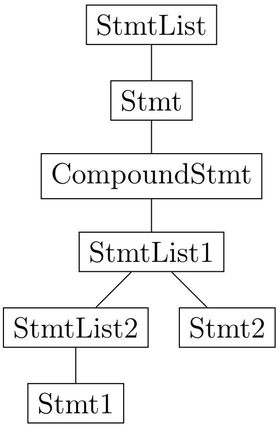
AST:



- Block (Compound) Statement List:

```
    {
      Stmt1;
      Stmt2;
    }
```

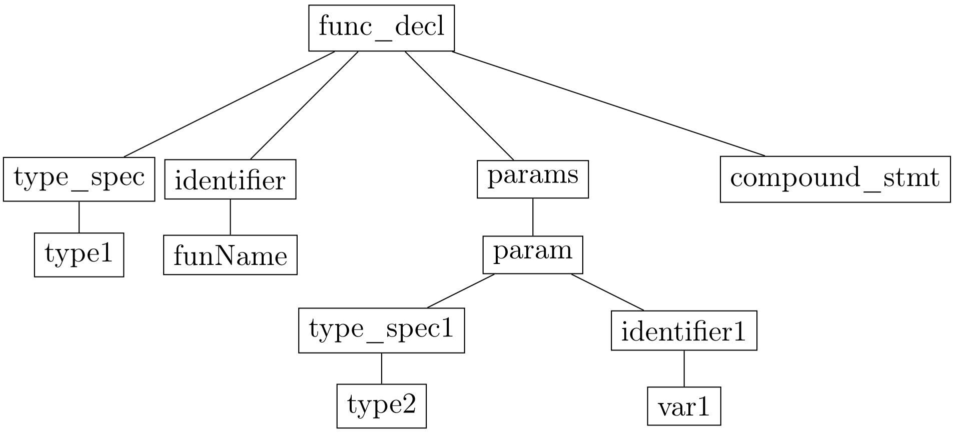
AST:



- Function Definition:

```
type1 funName (type2 var1)
{
        stmt;
}
```

AST:



Input:

The input to the assignment is a subset of C programs restricted to the grammar mentioned above.

Output:

Print the following in the same order:

- Longest Path of the Abstract Syntax Tree.
- Longest Path across all the subtrees of 'if' statement. If there are no if statements, Print '0'.
- Longest Path across all the subtrees of 'while' statement. If there are no while statements, Print '0'.
- Longest Path of the 'main' function subtree.

Note:

The following grammar is left-recursive. Please use right-recursive grammar wherever you encounter left recursion.

Grammar:

Use the following Grammar for the assignment:

```
       program → decl_list
     decl_list → decl_list decl | decl
          decl → var_decl | func_decl
      var_decl → type_spec identifier ";"
                 | type_spec identifier "," var_decl
                 | type_spec identifier "[" integerLit "]" ";"
                 | type_spec identifier "[" integerLit "]" "," var_decl
     type_spec → "void" | "int" | "float"
                 | "void" "*" | "int" "*" | "float" "*"
      fun_decl → type_spec identifier "(" params ")" compound_stmt
        params → param_list | ϵ
    param_list → param_list "," param | param
         param → type_spec identifier | type_spec "[" "]" identifier
     stmt_list → stmt_list stmt | stmt
          stmt → assign_stmt | compound_stmt | if_stmt | while_stmt
                 | return_stmt | break_stmt | continue_stmt
     expr_stmt → expr ";"
    while_stmt → "while" "(" expr ")" stmt
 compound_stmt → "{" local_decls stmt_list "}"
   local_decls → local_decls local_decl | ϵ
    local_decl → type_spec identifier ";"
                 | type_spec identifier "[" expr "]" ";"
       if_stmt → "if" "(" expr ")" stmt
                 | "if" "(" expr ")" stmt "else" stmt
   return_stmt → "return" ";" | "return" expr ";"
    break_stmt → "break" ";"
 continue_stmt → "continue" ";"
   assign_stmt → identifier "=" expr | identifier "[" expr "]" "=" expr
          expr → Pexpr "||" Pexpr
               → Pexpr "==" Pexpr | Pexpr "!=" Pexpr
               → Pexpr "<=" Pexpr | Pexpr "<" Pexpr | Pexpr ">=" Pexpr | Pexpr ">" Pexpr
               → Pexpr "&&" Pexpr
               → Pexpr "+" Pexpr | Pexpr "-" Pexpr
               → Pexpr "*" Pexpr | Pexpr "/" Pexpr | Pexpr "%" Pexpr
               → "!" Pexpr | "-" Pexpr | "+" Pexpr | "*" Pexp | "&" Pexp
               → Pexpr
               → identifier "(" args ")"
               → identifier "[" expr "]"
         Pexpr → integerLit | floatLit | identifier | "(" expr ")"
    integerLit → <INTEGER_LITERAL>
      floatLit → <FLOAT_LITERAL>
    identifier → <IDENTIFIER>
      arg_list → arg_list "," expr | expr
          args → arg_list | ϵ
```

Example:

Input:

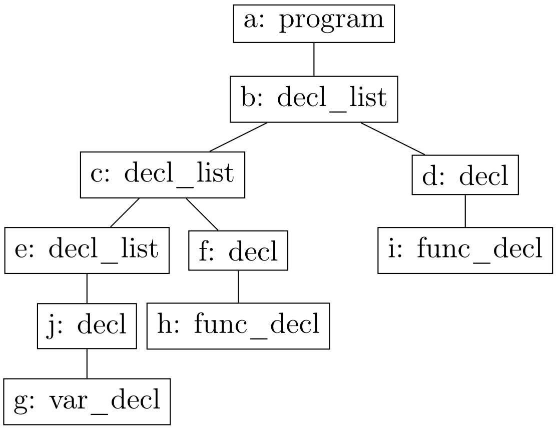
```
int d[10];
int foo(int c, int[] b)
{
    return b[c];
}

int main()
{
    int i;
    i = 0;

    if (i == 0)
      i = i + 1;

    while (i < 10)
    {
      i = i + 1;
      }

    return foo(4, d);
}
```

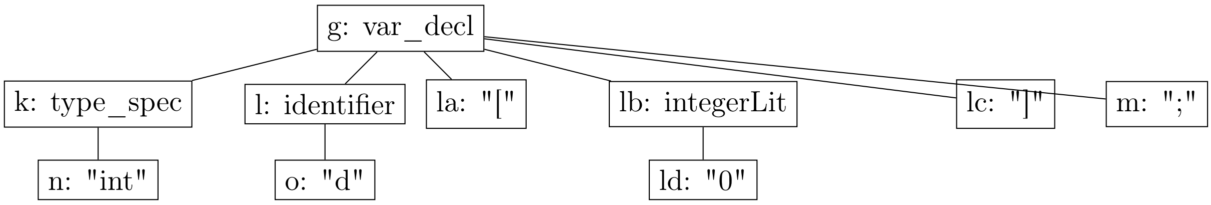
Output:

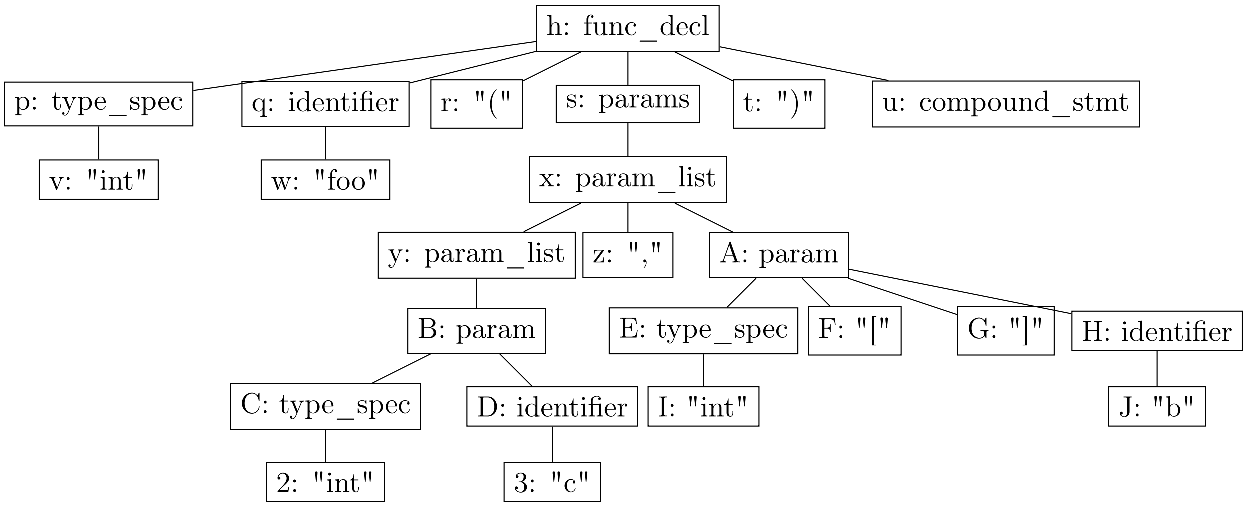
- 30 (Rooted at a)
- 14 (Rooted at 63)
- 17 (Rooted at 35)
- 16 (Rooted at i)

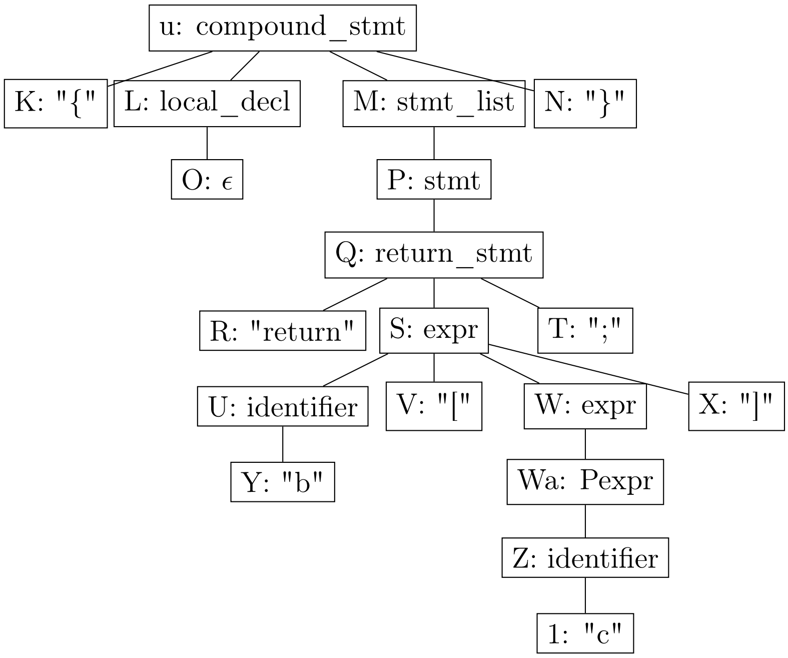
The actual output has to contain only one number on each line.

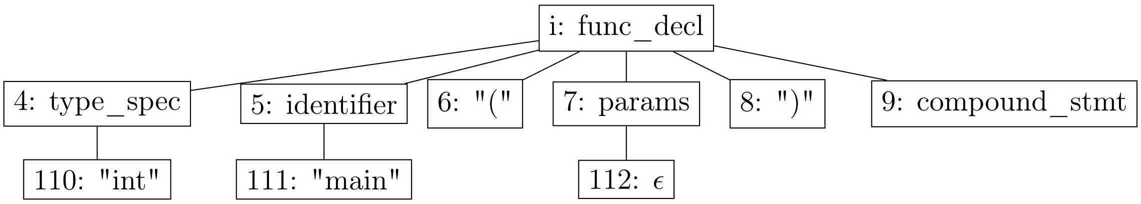
AST:

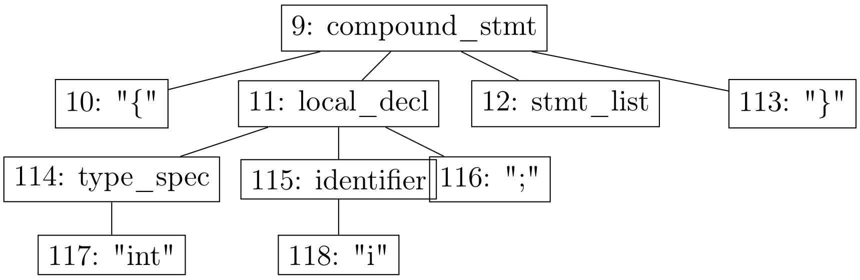

a: program
b: decl_list
c: decl_list
d: decl
e: decl_list
f: decl
i: func_decl
j: decl
h: func_decl
g: var_decl

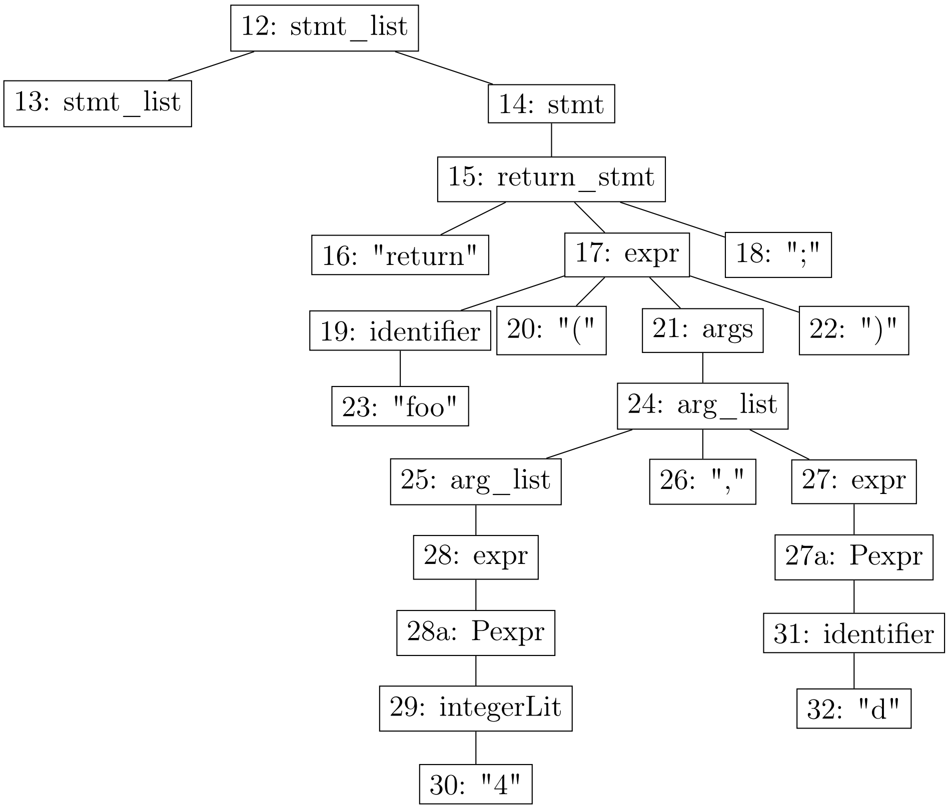

g: var_decl
k: type_spec
l: identifier
la: "["
lb: integerLit
lc: "]"
m: ";"
n: "int"
o: "d"
ld: "0"

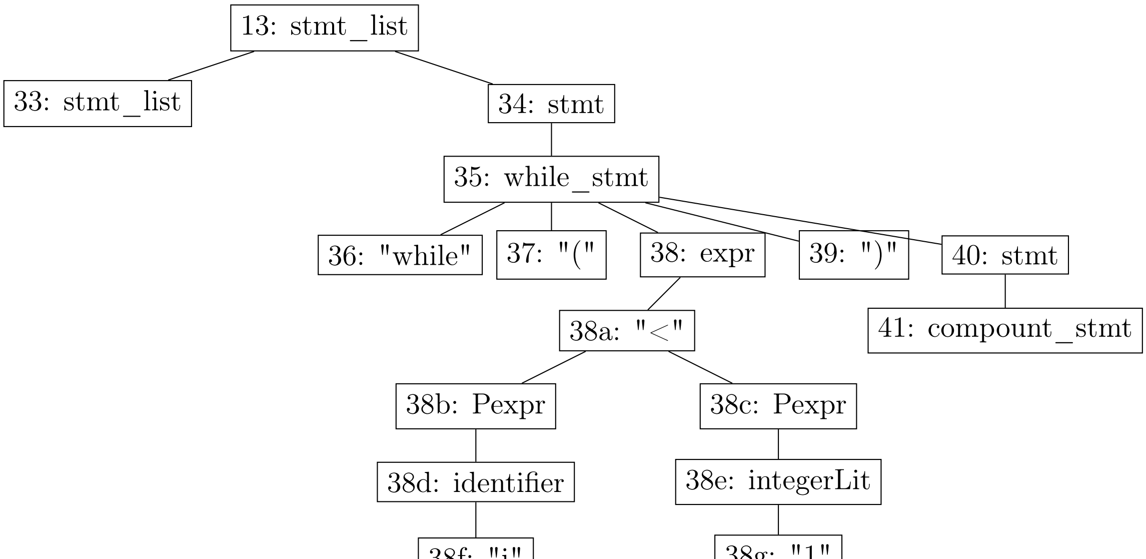

h: func_decl
p: type_spec
q: identifier
r: "("
s: params
t: ")"
u: compound_stmt
v: "int"
w: "foo"
x: param_list
y: param_list
z: ","
A: param
B: param
E: type_spec
F: "["
G: "]"
H: identifier
C: type_spec
D: identifier
I: "int"
J: "b"
2: "int"
3: "c"

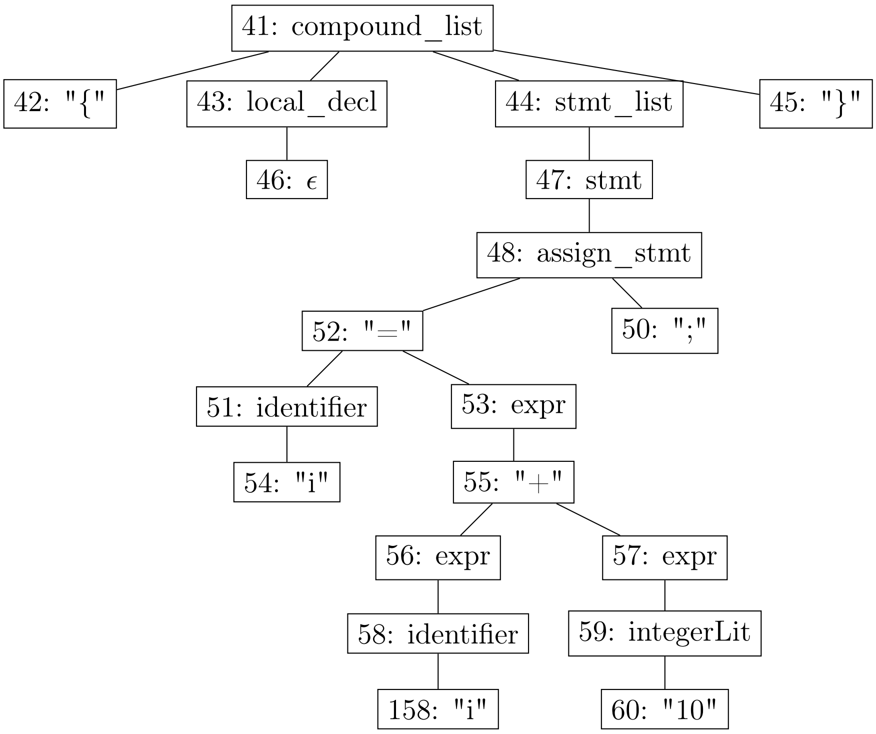

u: compound_stmt
K: "{"
L: local_decl
M: stmt_list
N: "}"
O: ϵ
P: stmt
Q: return_stmt
R: "return"
S: expr
T: ";"
U: identifier
V: "["
W: expr
X: "]"
Y: "b"
Wa: Pexpr
Z: identifier
1: "c"

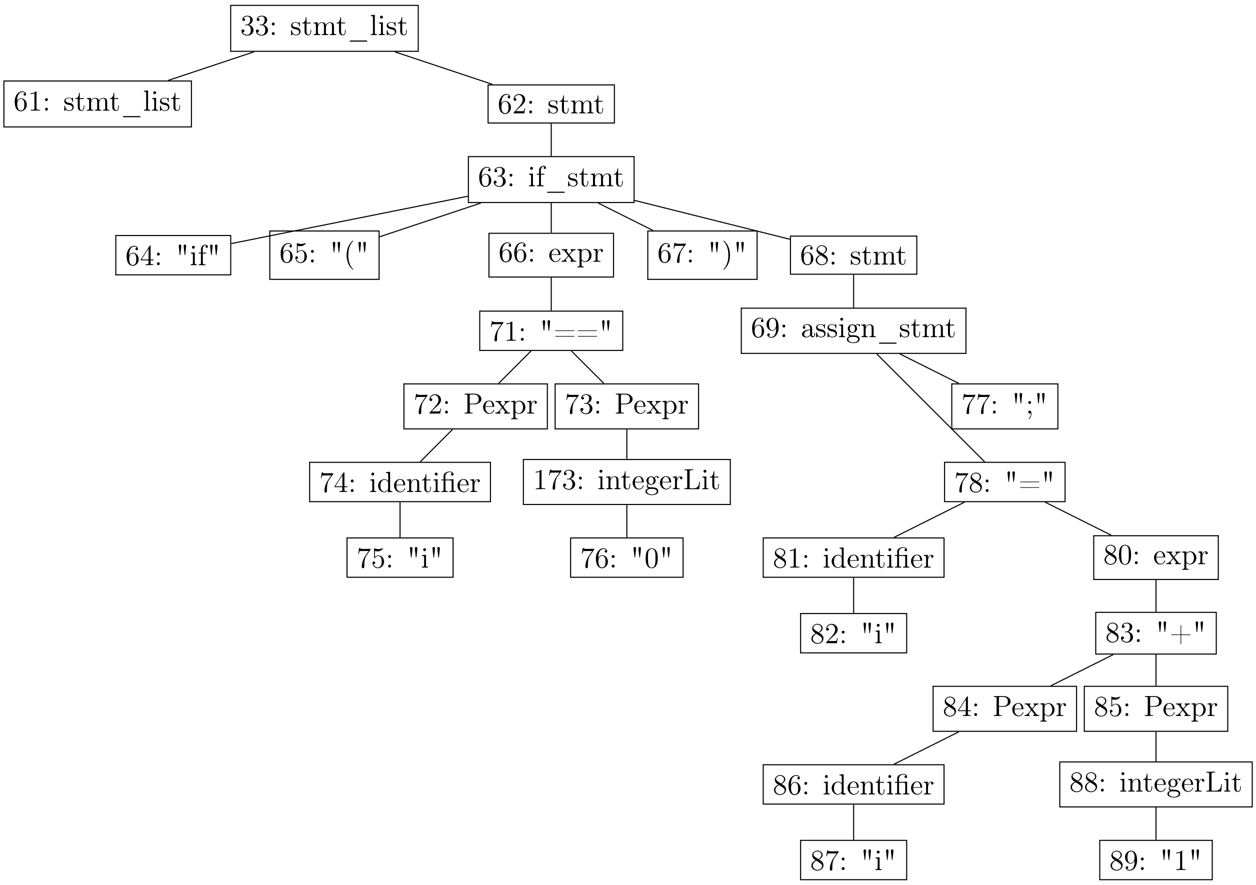

i: func_decl
4: type_spec
5: identifier
6: "("
7: params
8: ")"
9: compound_stmt
110: "int"
111: "main"
112: ϵ

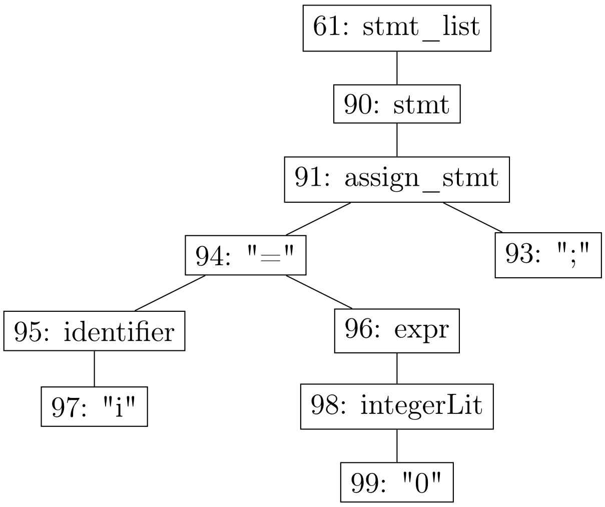

9: compound_stmt
10: "{"
11: local_decl
12: stmt_list
113: "}"
114: type_spec
115: identifier
116: ";"
117: "int"
118: "i"


12: stmt_list
13: stmt_list
14: stmt
15: return_stmt
16: "return"
17: expr
18: ";"
19: identifier
20: "("
21: args
22: ")"
23: "foo"
24: arg_list
25: arg_list
26: ","
27: expr
28: expr
27a: Pexpr
28a: Pexpr
31: identifier
29: integerLit
32: "d"
30: "4"


13: stmt_list
33: stmt_list
34: stmt
35: while_stmt
36: "while"
37: "("
38: expr
39: ")"
40: stmt
41: compount_stmt
38a: "<"
38b: Pexpr
38c: Pexpr
38d: identifier
38e: integerLit
38f: "i"
38g: "1"


41: compound_list
42: "{"
43: local_decl
44: stmt_list
45: "}"
46: ϵ
47: stmt
48: assign_stmt
52: "="
50: ";"
51: identifier
53: expr
54: "i"
55: "+"
56: expr
57: expr
58: identifier
59: integerLit
158: "i"
60: "10"


33: stmt_list
61: stmt_list
62: stmt
63: if_stmt
64: "if"
65: "("
66: expr
67: ")"
68: stmt
71: "=="
69: assign_stmt
72: Pexpr
73: Pexpr
77: ";"
74: identifier
173: integerLit
78: "="
75: "i"
76: "0"
81: identifier
80: expr
82: "i"
83: "+"
84: Pexpr
85: Pexpr
86: identifier
88: integerLit
87: "i"
89: "1"


61: stmt_list
90: stmt
91: assign_stmt
94: "="
93: ";"
95: identifier
96: expr
97: "i"
98: integerLit
99: "0"