

## Assignment #2

Abstract Syntax Tree

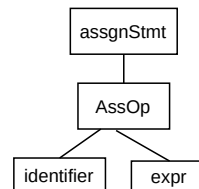
Deadline : 27/09/2020, 11:55PM

## 1 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.

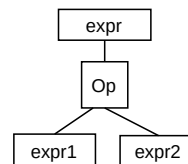
The Abstract Syntax Tree for various C constructs are as follows: (Note:  $\text{expr} < i >$  are instances of 'expr' numbered just for clarity.)

### • Assignment Statement

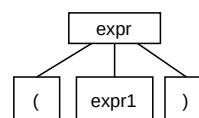
 $\text{assgnStmt} \rightarrow \text{var AssOp expr}$ 
 $\text{AssOp} \rightarrow \{ = \}$ 


### • Expressions

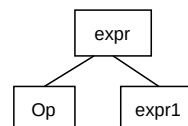
- Binary Operators

 $\text{expr} \rightarrow \text{expr1 Op expr2}$ 
 $\text{Op} \rightarrow \{ \text{Relational operators, Binary Operators, Logical Operators} \}$ 


- Parentheses expression

 $\text{expr} \rightarrow ( \text{expr1} )$ 


- Unary Operators

 $\text{expr} \rightarrow \text{Op expr1}$ 
 $\text{Op} \rightarrow \{ \text{Unary Operators} \}$ 


- **If Statement**

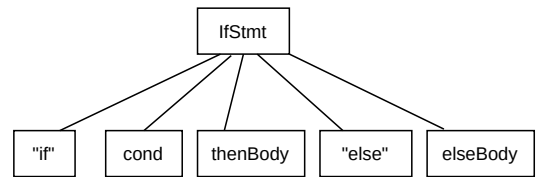
$$\text{If} \rightarrow \text{if (cond) } \{$$

$$\quad \text{thenBody}$$

$$\quad \} \text{ else } \{$$

$$\quad \text{elseBody}$$

$$\quad \}$$

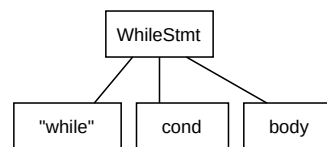


- **While Statement**

$$\text{WhileStmt} \rightarrow \text{while (cond) } \{$$

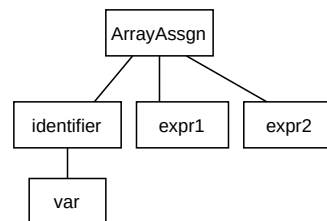
$$\quad \text{body}$$

$$\quad \}$$



- **Array Assignment Statement**

$$\text{ArrayAssgn} \rightarrow \text{var [ expr1 ] = expr2 ;}$$



- **Variable Declaration**

$$\text{VarDecl} \rightarrow \text{type var ;}$$

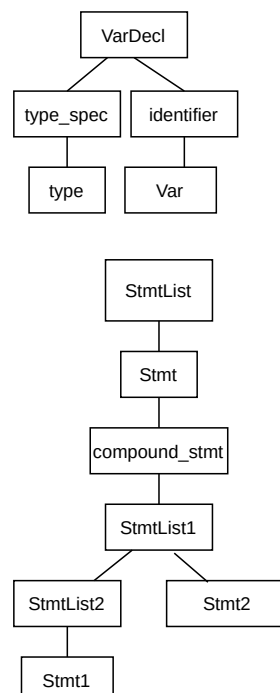
- **Block (Compound) Statement List**

$$\{$$

$$\text{Stmt1;}$$

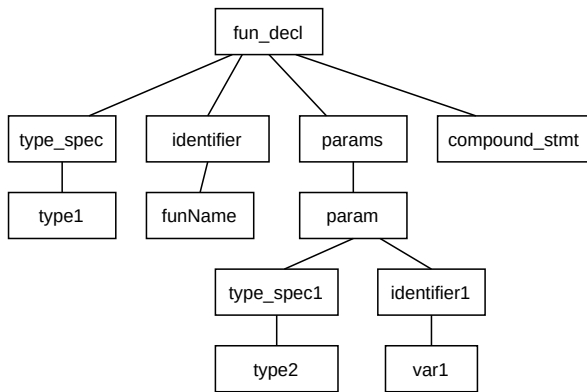
$$\text{Stmt2;}$$

$$\}$$



- **Function Definition**

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



## 2 Input

The input to the assignment is a subset of C programs restricted to the grammar mentioned in Section 4.

## 3 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 across all the subtrees of 'switch' statement. If there are no switch statements, Print '0'.
- Longest Path of the 'main' function subtree.

## 4 Grammar

Below is the grammar to be used for the assignment :

```
program → decl_list
decl_list → decl_list decl | decl
decl → var_decl | func_decl | struct_decl
struct_decl → "struct" identifier "{" local_decls "}" ";"
var_decl → type_spec identifier ";"
          | type_spec identifier "," var_decl
          | type_spec identifier "[" integerLit "]" ";"
          | type_spec identifier "[" integerLit "]" "," var_decl
type_spec → extern_spec "void" | extern_spec "int" | extern_spec "float"
          | extern_spec "void" "*" | extern_spec "int" "*" | extern_spec "float" "*"
          | "struct" identifier | "struct" identifier "*"
extern_spec → "extern" | ε
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 | switch_stmt
      | return_stmt | break_stmt | continue_stmt | dowhile_stmt | print_stmt
      | incr_stmt | decr_stmt
while_stmt → "while" "(" expr ")" stmt
dowhile_stmt → "do" stmt "while" "(" "expr" ")" ";"
print_stmt → "printf" "(" format_specifier "," identifier ")" ";"
format_specifier → "%d"
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" ";"
switch_stmt → "switch" "(" expr ")" "{" compound_case default_case "}"
compound_case → single_case compound_case
               | single_case
single_case → "case" integerLit ":" stmt_list
default_case → "default" ":" stmt_list
```

$\text{assign\_stmt} \rightarrow \text{identifier} \text{ "=" expr ";" } \mid \text{identifier} \text{ "[" expr "]" "=" expr ";" }$   
 $\quad \mid \text{identifier} \text{ "->" identifier "=" expr ";" }$   
 $\quad \mid \text{identifier} \text{ "." identifier "=" expr ";" }$   
 $\text{incr\_stmt} \rightarrow \text{identifier} \text{ "++" ";" }$   
 $\text{decr\_stmt} \rightarrow \text{identifier} \text{ "--" ";" }$   
 $\text{expr} \rightarrow \text{Pexpr} \text{ "<" Pexpr } \mid \text{Pexpr} \text{ ">" Pexpr }$   
 $\quad \rightarrow \text{Pexpr} \text{ "<=" Pexpr } \mid \text{Pexpr} \text{ ">=" Pexpr }$   
 $\quad \rightarrow \text{Pexpr} \text{ "||" Pexpr } \mid \text{"sizeof" "(" Pexpr ")"}$   
 $\quad \rightarrow \text{Pexpr} \text{ "==" Pexpr } \mid \text{Pexpr} \text{ "!=" Pexpr } \mid \text{Pexpr} \text{ "<=>" Pexpr }$   
 $\quad \rightarrow \text{Pexpr} \text{ "&\&" Pexpr } \mid \text{Pexpr} \text{ "->" Pexpr }$   
 $\quad \rightarrow \text{Pexpr} \text{ "+" Pexpr } \mid \text{Pexpr} \text{ "-" Pexpr }$   
 $\quad \rightarrow \text{Pexpr} \text{ "*" Pexpr } \mid \text{Pexpr} \text{ "/" Pexpr } \mid \text{Pexpr} \text{ "%" Pexpr }$   
 $\quad \rightarrow \text{"!" Pexpr } \mid \text{"-" Pexpr } \mid \text{"+" Pexpr } \mid \text{"*" Pexp } \mid \text{"\&" Pexp }$   
 $\quad \rightarrow \text{Pexpr}$   
 $\quad \rightarrow \text{identifier} \text{ "(" args ")"}$   
 $\quad \rightarrow \text{identifier} \text{ "[" expr "]"}$   
 $\text{Pexpr} \rightarrow \text{integerLit} \mid \text{floatLit} \mid \text{identifier} \mid \text{"(" expr ")"}$   
 $\text{integerLit} \rightarrow \text{< INTEGER\_LITERAL >}$   
 $\text{floatLit} \rightarrow \text{< FLOAT\_LITERAL >}$   
 $\text{identifier} \rightarrow \text{< IDENTIFIER >}$   
 $\text{arg\_list} \rightarrow \text{arg\_list} \text{ "," expr } \mid \text{expr}$   
 $\text{args} \rightarrow \text{arg\_list} \mid \epsilon$

## 5 Example

### 5.1 Input

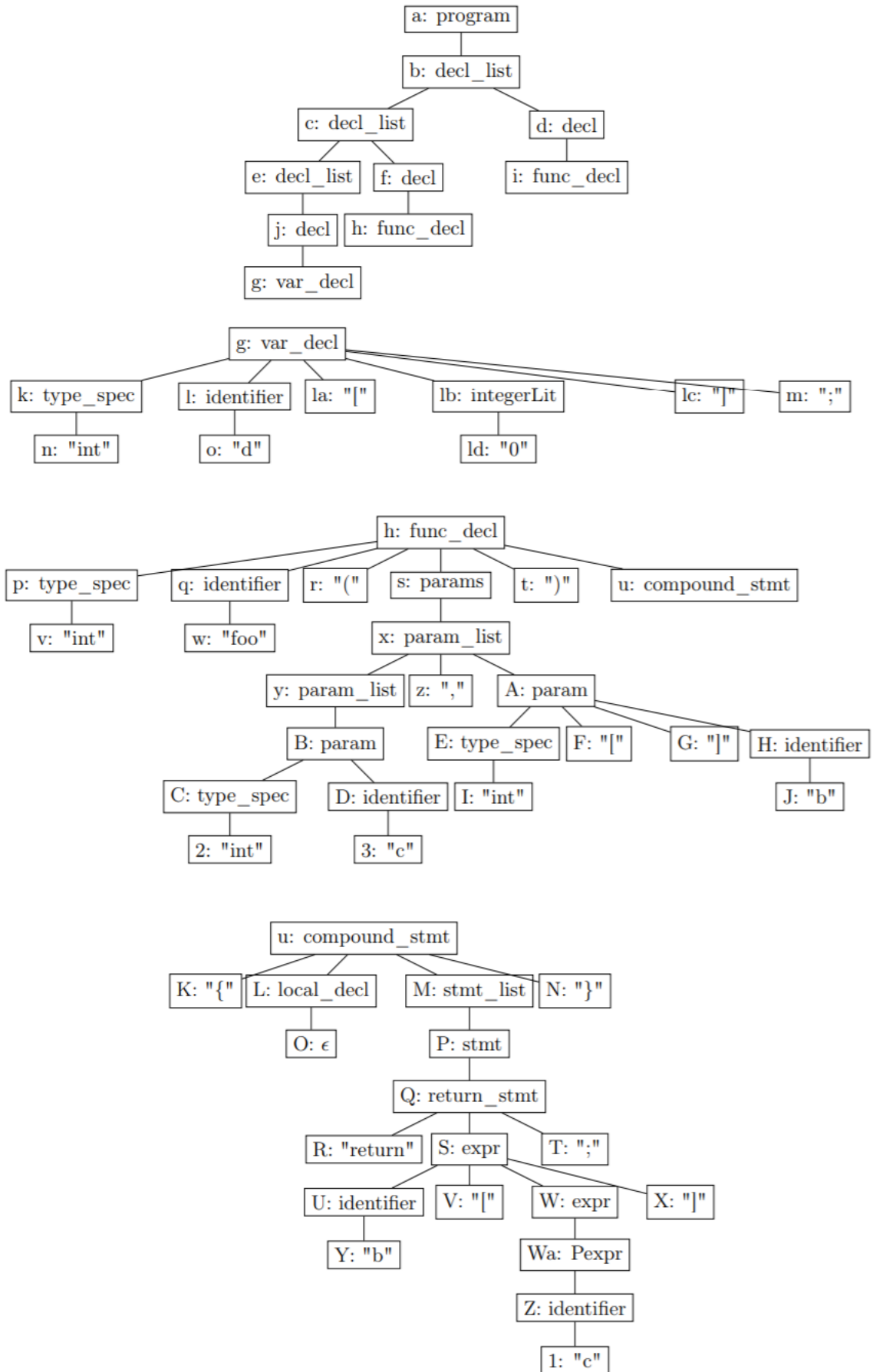
```
int d [ 10 ] ;
int foo ( int c , int [] b)
{
    return b [ c ] ;
}
int main ( )
{
    int i ;
    i = 0;
    if ( i == 0 )
    i = i + 1;
    while ( i < 10)
    {
        i = i + 1;
    }
    return foo (4 , d);
}
```

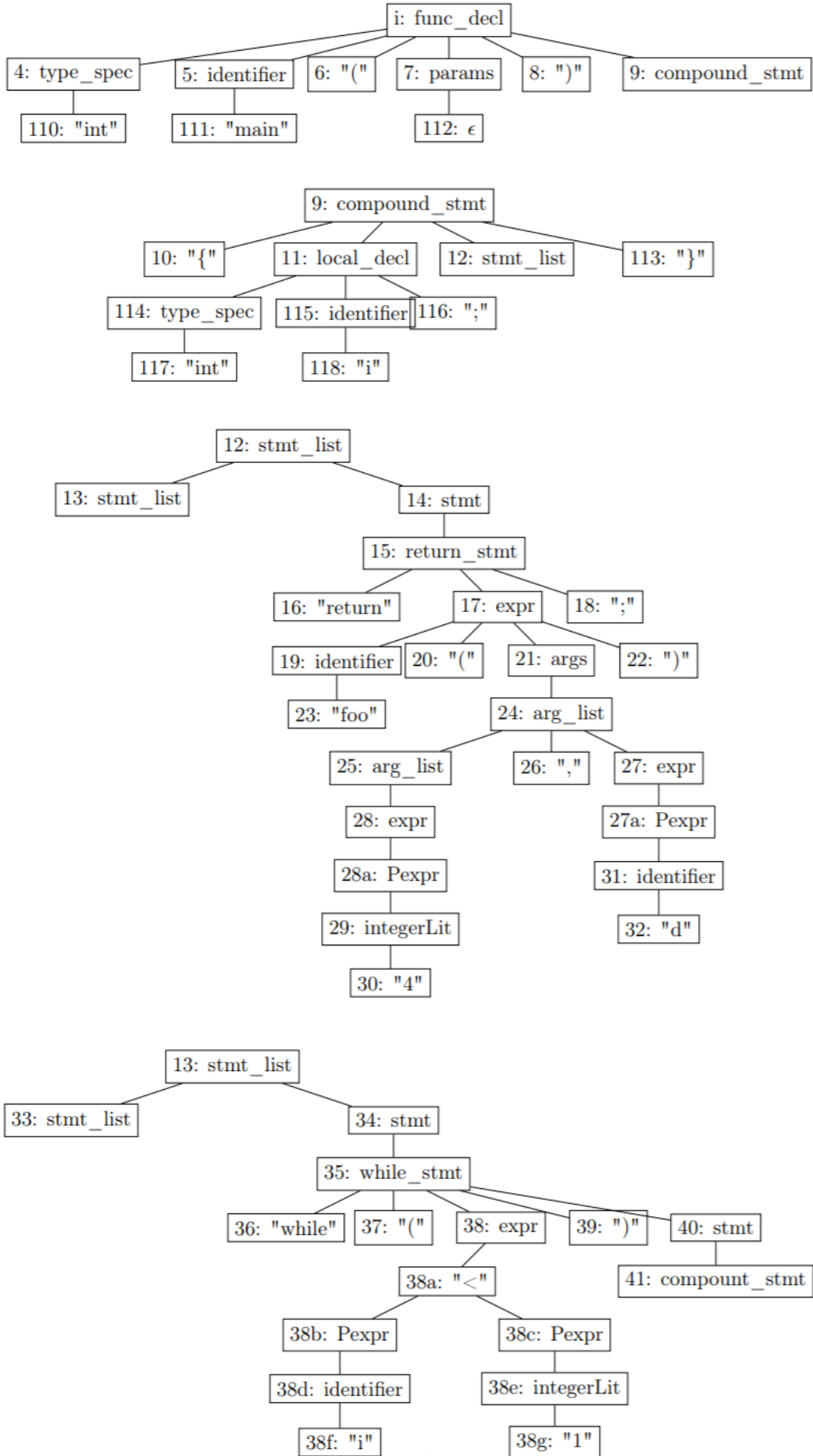
### 5.2 Output

- 31 (Rooted at a)
- 14 (Rooted at 63)
- 17 (Rooted at 35)
- 0
- 25 (Rooted at i)

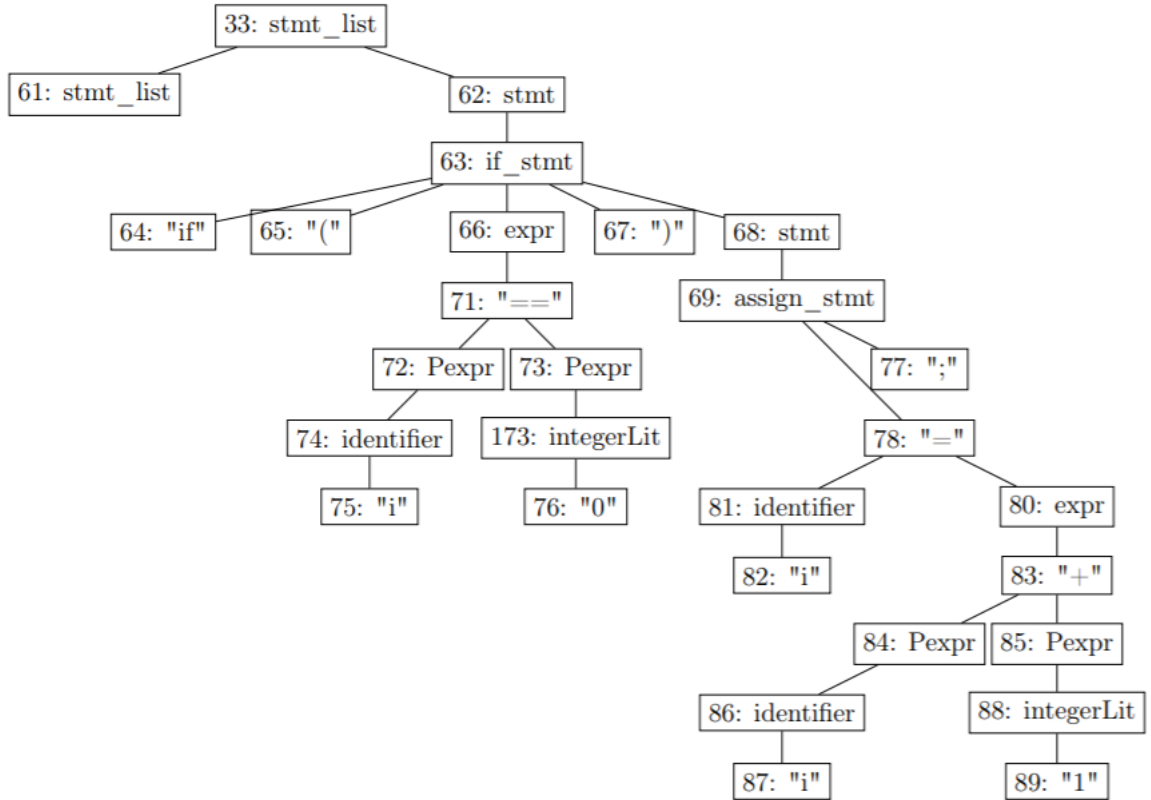
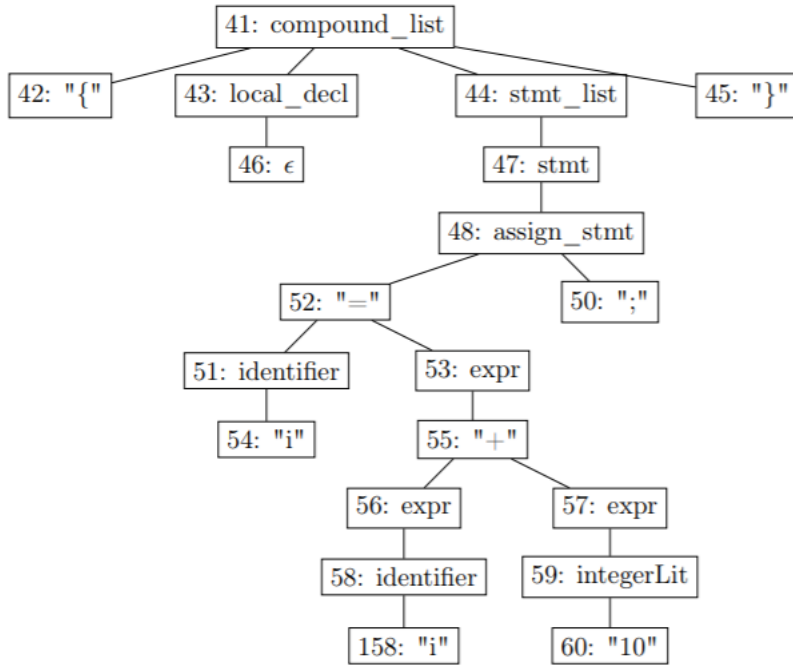
The actual output has to contain only one number on each line.

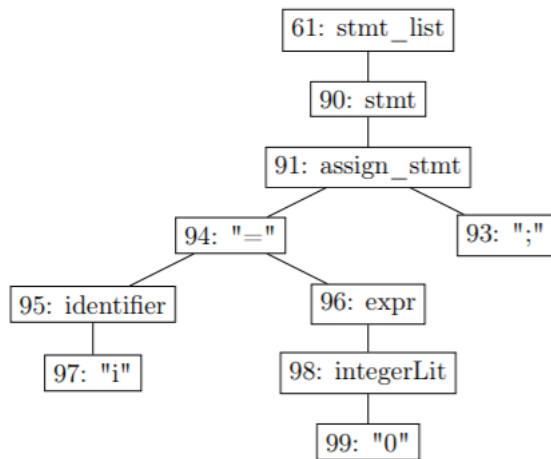
### 5.3 AST :











## 6 Submission

Submit a tar.gz file with filename as <ROLLNO>\_A2.tar.gz (eg. CS15B001\_A2.tar.gz) having the following structure:

- CS15B001 <directory>
  - \*.l
  - \*.y
  - Makefile

**Note:** The Makefile should run lex, yacc, compile the generated code and generate an executable a.out file.