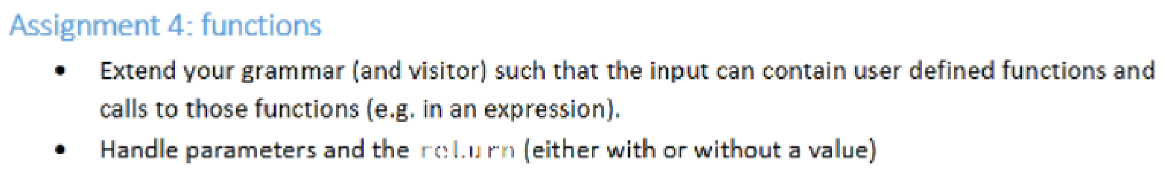
Week 4- AUT Assignment

Group 3

Table of contents

Assignment 4

Problem description



Assignment 4: functions

- Extend your grammar (and visitor) such that the input can contain user defined functions and calls to those functions (e.g. in an expression).
- Handle parameters and the return (either with or without a value)

Implementation

For this week, we had to extend our grammar to add support for functions so that the user can declare and call functions. Therefore, we have added 3 new rules in our grammar:

- function_declaration
- function_call
- returnStat

While testing our code, we found some weird behavior when we have multiple return statements. For example, consider the following code:

```
int m = 10
function myFunction(int a){
    if(a > 0){
        print m
        return a + m
    }
   return a
}
int c = myFunction(1)
```

Since we have multiple return statements, the compiler always executes all the statements. In order to fix this, we had to check if there is an if_statement and a return_statement in the same code_block, so that we tell the compiler to execute the line and then stop. If there is no such statement, then just continue visiting the other statements.

```
grammar Example2;

start2    : statement* EOF;

statement:  print_func
        |   while_statement
        |   returnStat
        |   if_statement
        |   function_declaration
        |   function_call
        |   expression
        |   variables

        ;

print_func:   Print op=(INT| BOOLEAN |ID |STRING)  #printVar
         |   Print mathExpression                  #printExpr
         ;

variables : int_variable
          | bool_variable
         | string_variable
         ;


returnStat: RETURN expression;

variables_type: IntType| BoolType | StringType;

parameters_funcDec:  (variables_type ID (COMMA variables_type ID)*)? ;

parameters_funcCall:  (expression (COMMA expression)*)? ;

function_declaration : FUNCTION ID function_block;

function_block : PARANL parameters_funcDec PARANR code_block;

function_call : ID PARANL parameters_funcCall PARANR;

if_statement:  IF condition_block (ELSE code_block)?;

while_statement: WHILE condition_block;
```

```
condition_block: PARANL expression PARANR code_block;

code_block:   OPEN_CURLY_BRACKET statement* CLOSE_CURLY_BRACKET
         |   statement
         ;

string_variable : StringType ID (IS_EQUAL expression)? # stringAssign
                | ID IS_EQUAL expression  # stringAssignValue
                ;

bool_variable : BoolType ID (IS_EQUAL expression)?  # boolAssign
              | ID IS_EQUAL expression  # boolAssignValue
              ;

int_variable :  IntType ID (IS_EQUAL mathExpression)?        # intAssign
             |  ID IS_EQUAL mathExpression                   #
intAssignValue
             |  ID (ADD_INCREMENT| SUB_INCREMENT | INCREMENT | DECREMENT)
mathExpression?  # incrementAndDecrementInt
             ;

expression : function_call  # FUNCTIONExpr
           |   mathExpression #MathExp
           |   BOOLEAN        #ValueBoolean
           |   STRING         #ValueString
           |   expression (GREATER_OR_EQUAL | SMALLER_OR_EQUAL |
GREATHER_THAN | SMALLER_THAN | EQUAL | NOT_EQUAL) expression   #
ComparisonExpression
           |   expression AND expression                            #
AndExpression
           |   expression OR expression                         #
OrExpression
           ;

mathExpression:  function_call                        # FUNCTIONMathExpr
            |   mathExpression op=MUL mathExpression #  Mul
            |   mathExpression op=DIV mathExpression #  Div
            |   mathExpression op=ADD mathExpression #  Add
            |   mathExpression op=SUB mathExpression #  Sub
            |   mathExpression op=POW mathExpression #  Pow
            |   mathExpression op=FACT  #  Fact
            |   PARANL mathExpression PARANR  # parens
            |   INT #ValueNumber
```

```
             |    ID    #ValueVariable
             ;


// tokens
IS_EQUAL: '=';
MUL:    '*';
DIV:    '/';
ADD:    '+';
SUB:    '-';
POW:    '^';
FACT:   '!';
PARANL: '(';
PARANR: ')';
DOT : '.';

ADD_INCREMENT : '+=';
SUB_INCREMENT : '-=';
INCREMENT : '++';
DECREMENT : '--';

OPEN_CURLY_BRACKET : '{';
CLOSE_CURLY_BRACKET : '}';

AND : 'and';
OR : 'or';

GREATER_OR_EQUAL : '>=';
SMALLER_OR_EQUAL : '<=';
GREATHER_THAN : '>';
SMALLER_THAN : '<';
EQUAL : '==';
NOT_EQUAL : '!=';


IntType: 'int';
BoolType: 'bool';
StringType: 'string';

WHILE : 'while';
COMMA: ',';
SEMICOLON: ';';
```

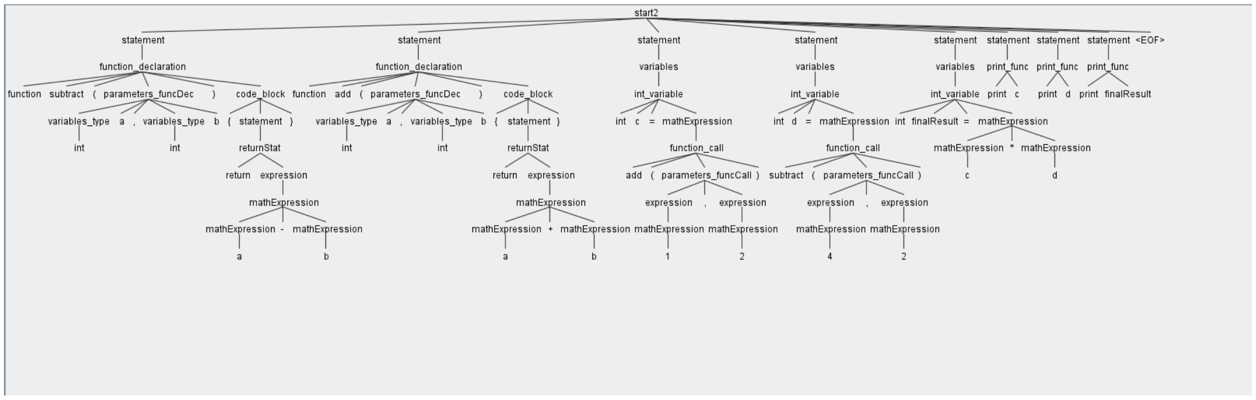
```
IF : 'if';
ELSE : 'else';
FOR : 'for';

VOID : 'void';
RETURN: 'return';
FUNCTION: 'function';
Print: 'print';

INT     : SUB?[0-9]+(DOT[0-9]+)? ;
BOOLEAN: 'True'|'False';
ID: [_A-Za-z][A-Za-z_!0-9.]* ;
STRING : '"' ~('\r' | '\n' | '"')* '"';

COMMENT : '//' .+? ('\n'|EOF) -> skip;
WS  : [ \t\r\n]+ -> skip;
```

Output

Below some test cases can be found to check the correctness of our grammar and implementations.

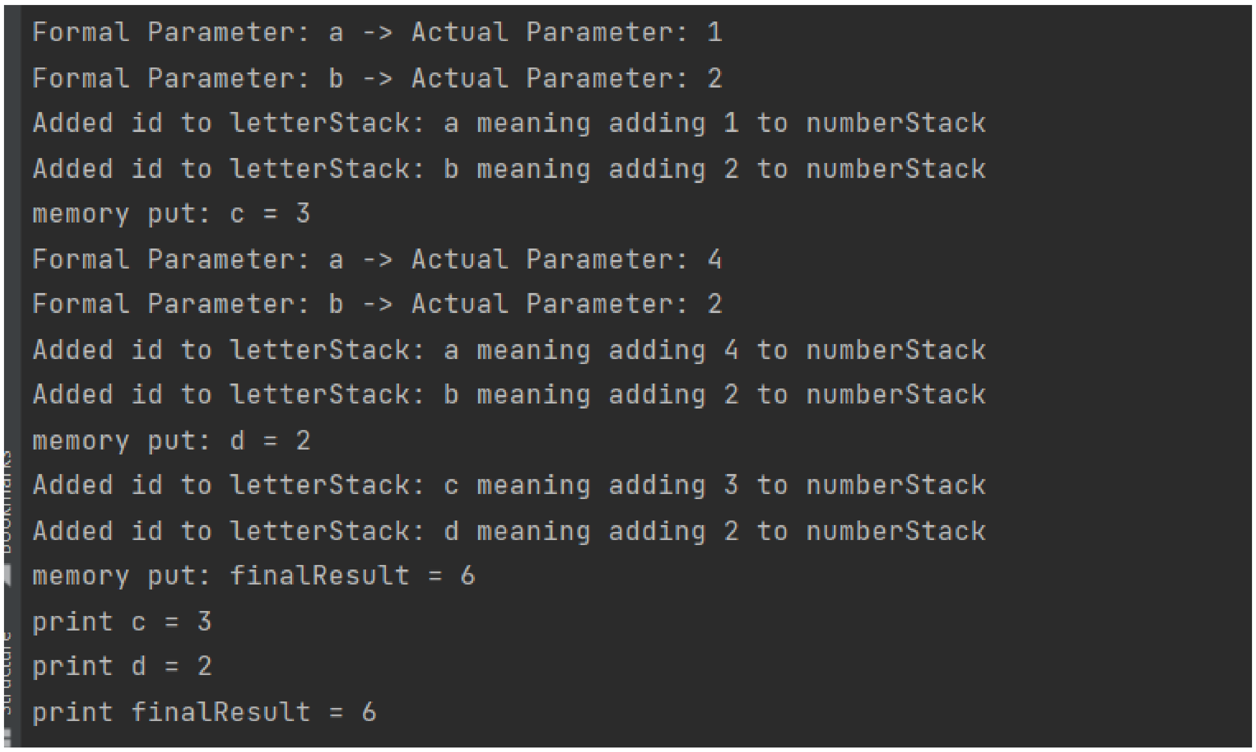
```
1) function subtract(int a, int b)
   {
     return a + b
   }
   function add (int a, int b)
   {
     return a - b
   }

   int c = add(1,2)
   int d = subtract(4,2)

   int finalResult = c + d

   print c
   print d
   print finalResult
```

Parse Tree



start2
statement statement statement statement statement statement statement statement <EOF>
function_declaration function_declaration variables variables variables print_func print_func print_func
function subtract (parameters_funcDec) code_block function add (parameters_funcDec) code_block int_variable int_variable int_variable print c print d print finalResult
variables_type a , variables_type b { statement } variables_type a , variables_type b { statement } int c = mathExpression int d = mathExpression int finalResult = mathExpression
int int returnStat int int returnStat function_call function_call mathExpression * mathExpression
return expression return expression add (parameters_funcCall) subtract (parameters_funcCall) c d
mathExpression mathExpression expression , expression expression , expression
mathExpression - mathExpression mathExpression + mathExpression mathExpression mathExpression mathExpression mathExpression
a b a b 1 2 4 2

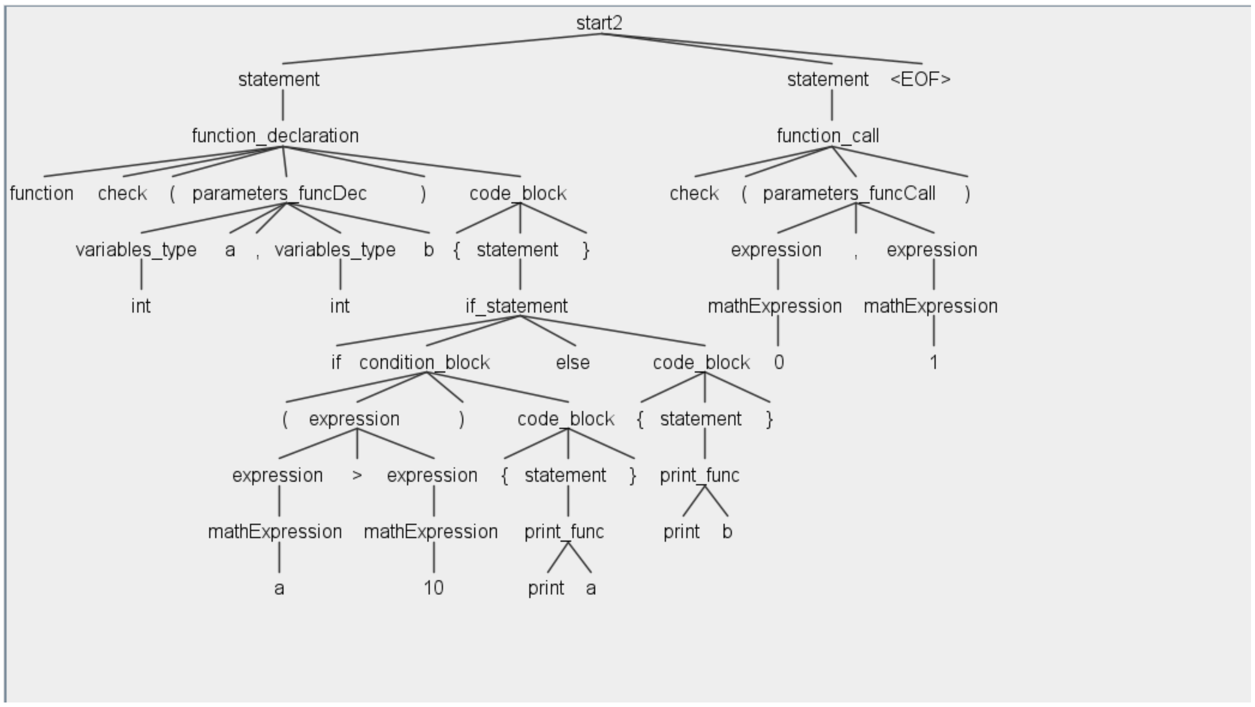
Terminal



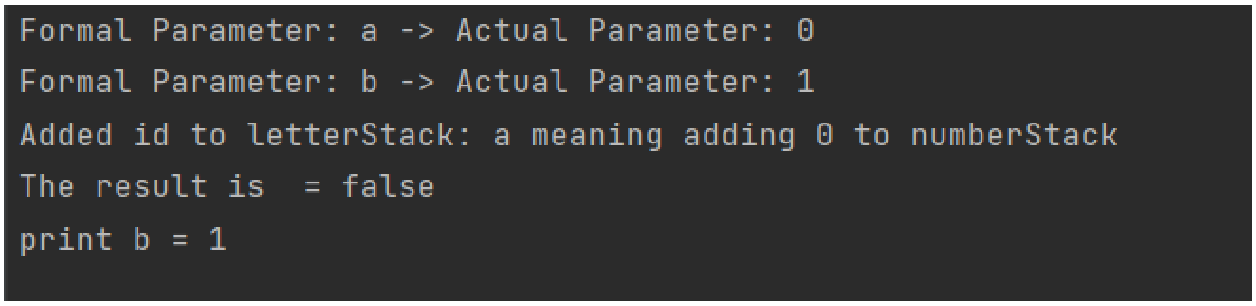
```
Formal Parameter: a -> Actual Parameter: 1
Formal Parameter: b -> Actual Parameter: 2
Added id to letterStack: a meaning adding 1 to numberStack
Added id to letterStack: b meaning adding 2 to numberStack
memory put: c = 3
Formal Parameter: a -> Actual Parameter: 4
Formal Parameter: b -> Actual Parameter: 2
Added id to letterStack: a meaning adding 4 to numberStack
Added id to letterStack: b meaning adding 2 to numberStack
memory put: d = 2
Added id to letterStack: c meaning adding 3 to numberStack
Added id to letterStack: d meaning adding 2 to numberStack
memory put: finalResult = 6
print c = 3
print d = 2
print finalResult = 6
```

```
2) function check(int a, int b){
       if(a > 10){
           print a
       }
       else{
          print b
       }
   }
   check(0, 1)
```

Parse Tree



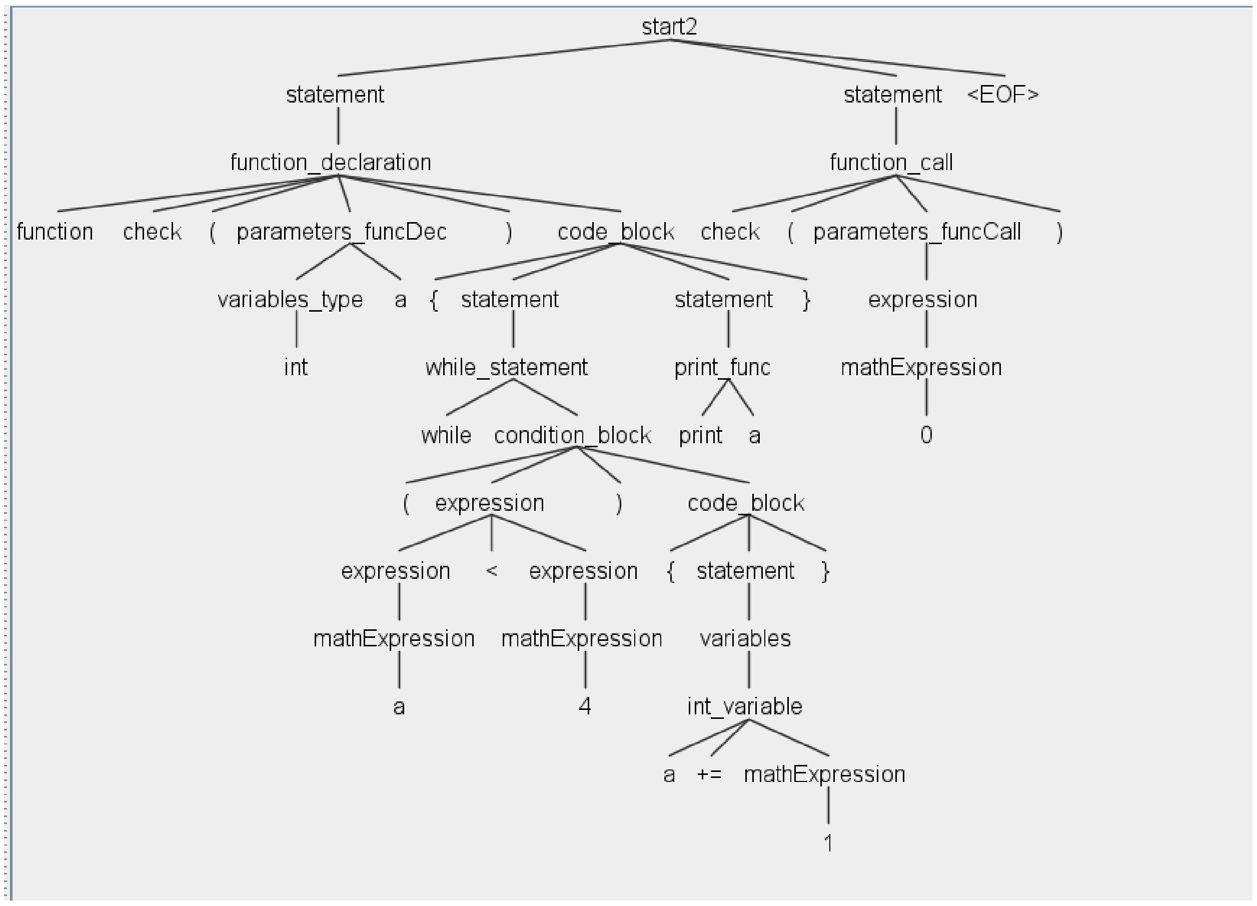
Terminal



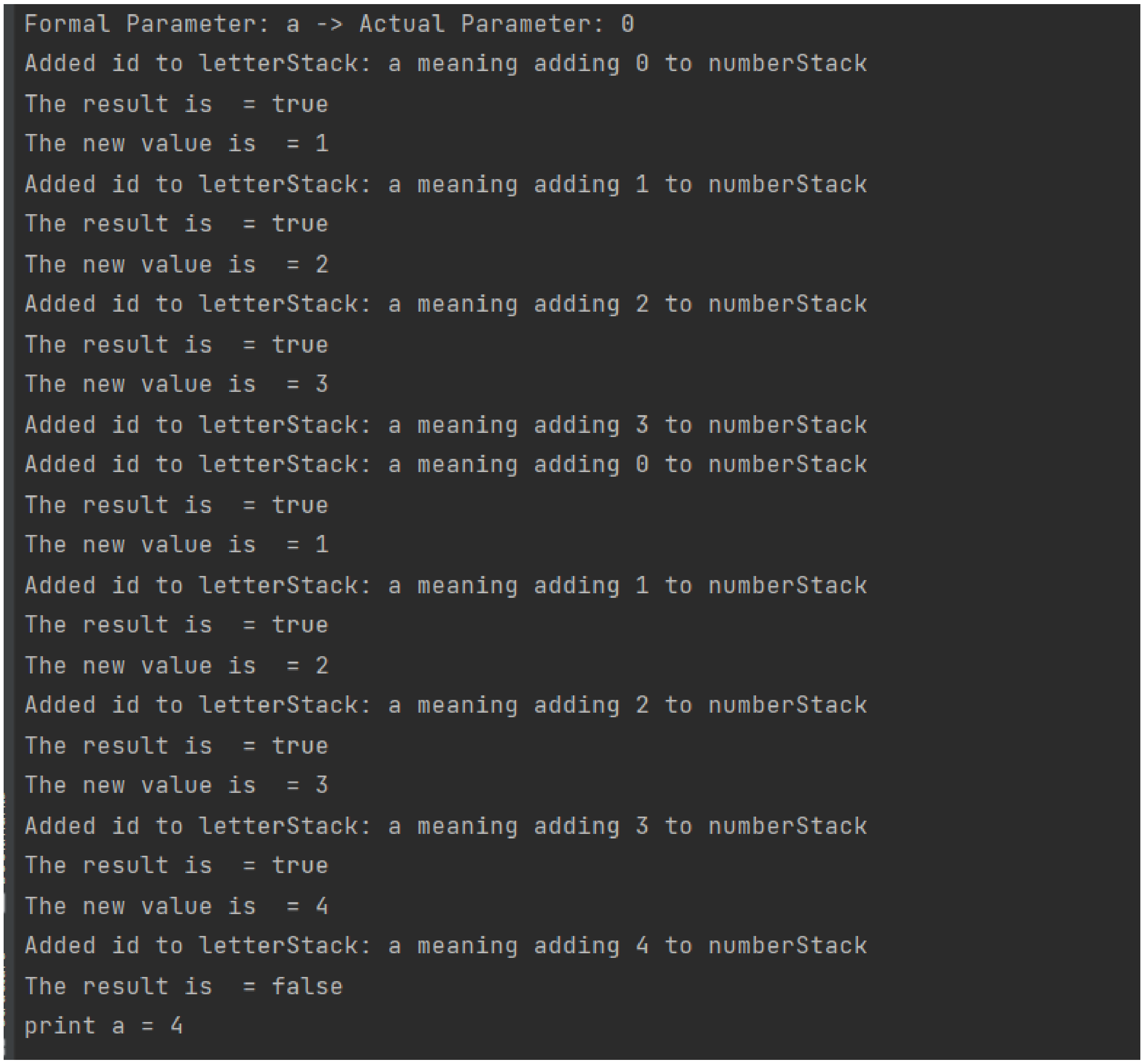
```
Formal Parameter: a -> Actual Parameter: 0
Formal Parameter: b -> Actual Parameter: 1
Added id to letterStack: a meaning adding 0 to numberStack
The result is  = false
print b = 1
```

```
3) function check(int a){
       while(a < 4){
          a += 1
       }
       print a
   }
   check(0)
```

Parse Tree



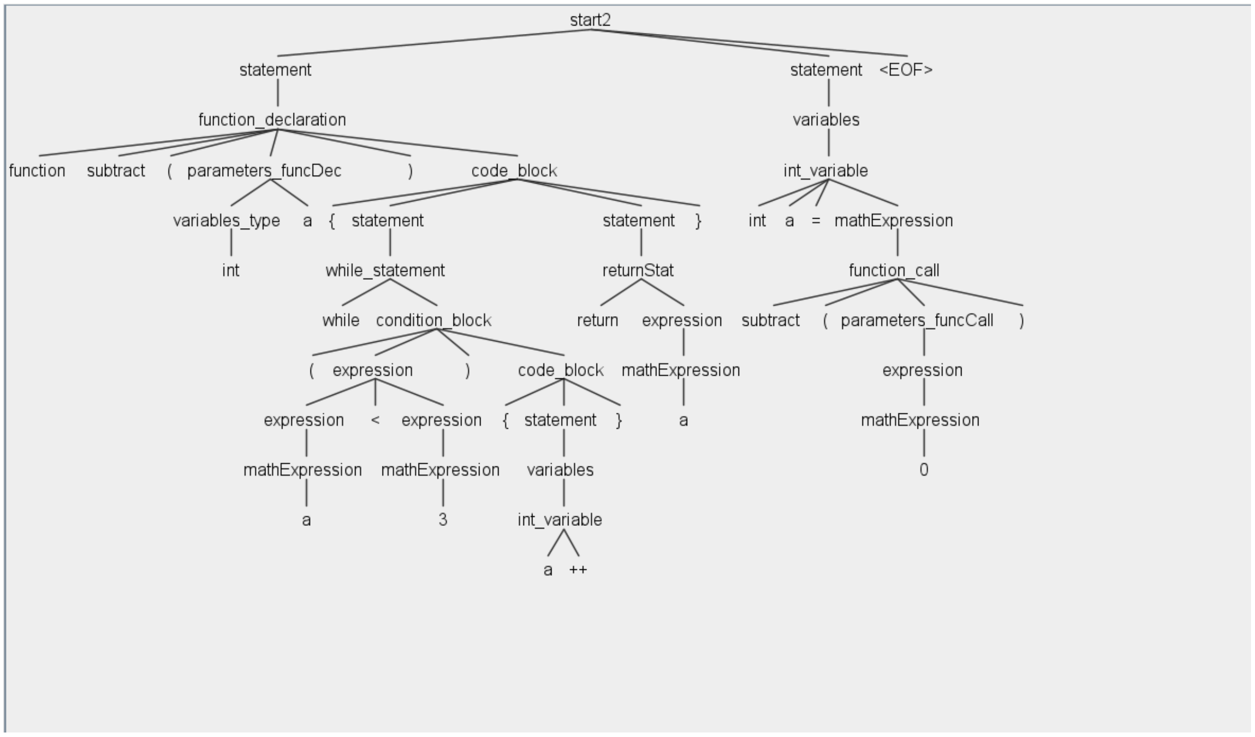
Terminal



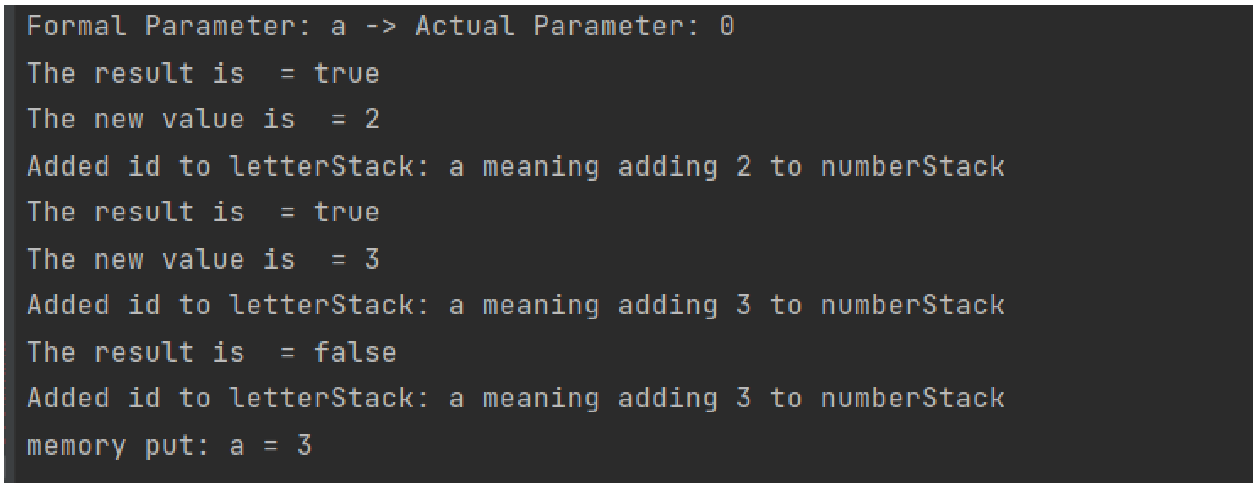
```
Formal Parameter: a -> Actual Parameter: 0
Added id to letterStack: a meaning adding 0 to numberStack
The result is  = true
The new value is  = 1
Added id to letterStack: a meaning adding 1 to numberStack
The result is  = true
The new value is  = 2
Added id to letterStack: a meaning adding 2 to numberStack
The result is  = true
The new value is  = 3
Added id to letterStack: a meaning adding 3 to numberStack
Added id to letterStack: a meaning adding 0 to numberStack
The result is  = true
The new value is  = 1
Added id to letterStack: a meaning adding 1 to numberStack
The result is  = true
The new value is  = 2
Added id to letterStack: a meaning adding 2 to numberStack
The result is  = true
The new value is  = 3
Added id to letterStack: a meaning adding 3 to numberStack
The result is  = true
The new value is  = 4
Added id to letterStack: a meaning adding 4 to numberStack
The result is  = false
print a = 4
```

```
4) function subtract(int a){
       while(a < 3){
           a++
       }
       return a
   }
   int a = subtract(0)
```

Parse Tree



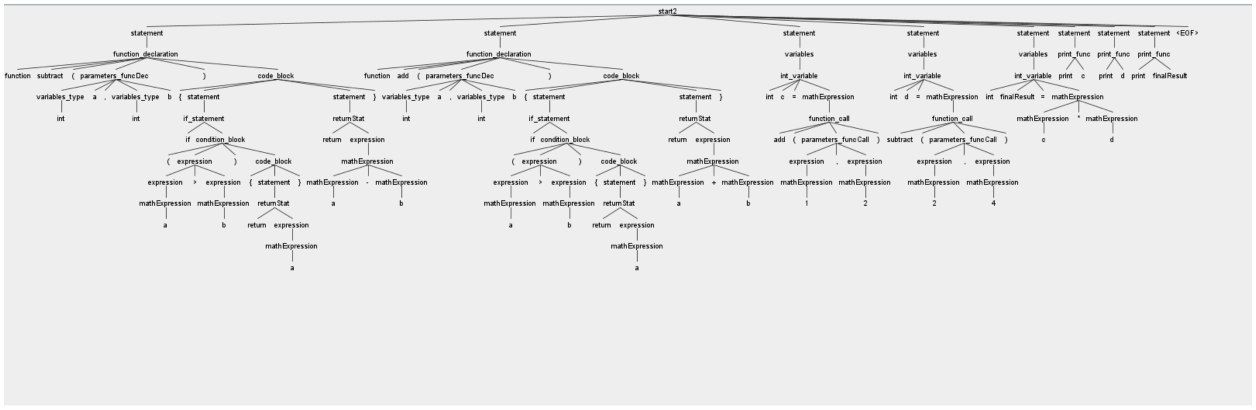
Terminal



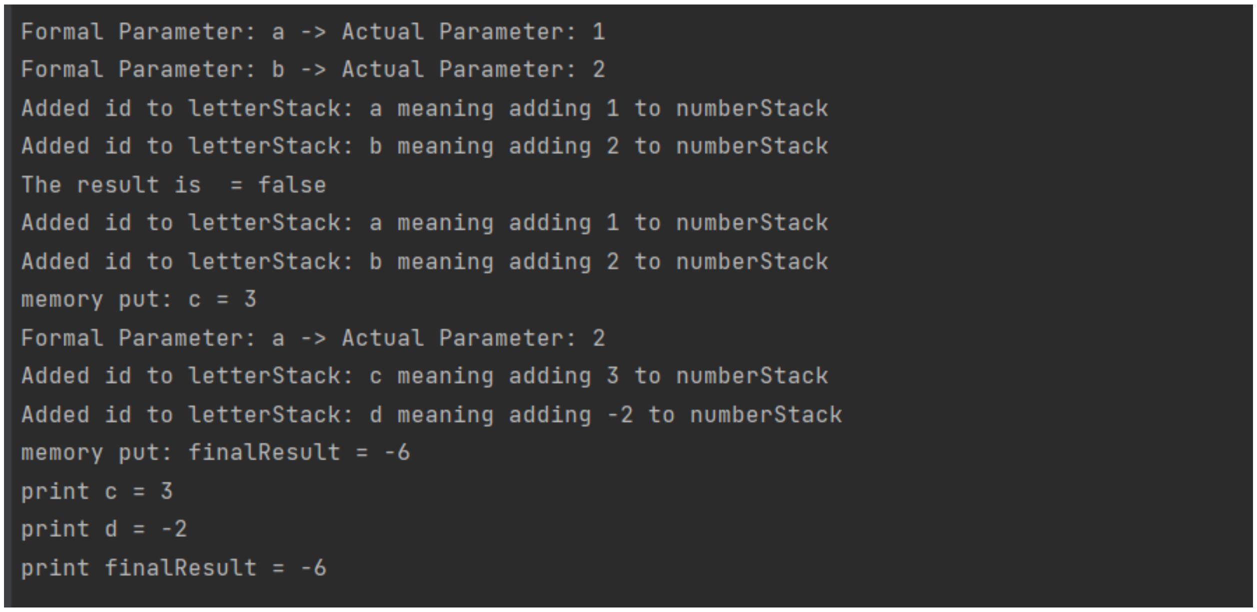
```
Formal Parameter: a -> Actual Parameter: 0
The result is  = true
The new value is  = 2
Added id to letterStack: a meaning adding 2 to numberStack
The result is  = true
The new value is  = 3
Added id to letterStack: a meaning adding 3 to numberStack
The result is  = false
Added id to letterStack: a meaning adding 3 to numberStack
memory put: a = 3
```

```
5) function subtract(int a, int b){
       if(a > b){
          return a
       }
       return a - b
   }
   function add(int a, int b)
   {
       if(a > b){
          return a
       }
       return a+b
   }
   int c = add(1,2)
   int d = subtract(2,4)
   int finalResult = c*d
   print c
   print d
   print finalResult
```

Parse Tree


start2
statement statement statement statement statement statement statement statement <EOF>
function_declaration function_declaration variables variables variables print_func print_func print_func
function subtract (parameters_funcDec) code_block function add (parameters_funcDec) code_block int_variable int_variable int_variable print c print d print finalResult
variables_type a , variables_type b { statement statement } variables_type a , variables_type b { statement statement } int c = mathExpression int d = mathExpression int finalResult = mathExpression
int int if_statement returnStat int int if_statement returnStat function_call function_call mathExpression * mathExpression
if condition_block return expression if condition_block return expression add (parameters_funcCall) subtract (parameters_funcCall) c d
(expression) code_block mathExpression (expression) code_block mathExpression expression , expression expression , expression
expression > expression { statement } mathExpression - mathExpression expression > expression { statement } mathExpression + mathExpression mathExpression mathExpression mathExpression mathExpression
mathExpression mathExpression returnStat a b mathExpression mathExpression returnStat a b 1 2 2 4
a b return expression a b return expression
mathExpression mathExpression
a a

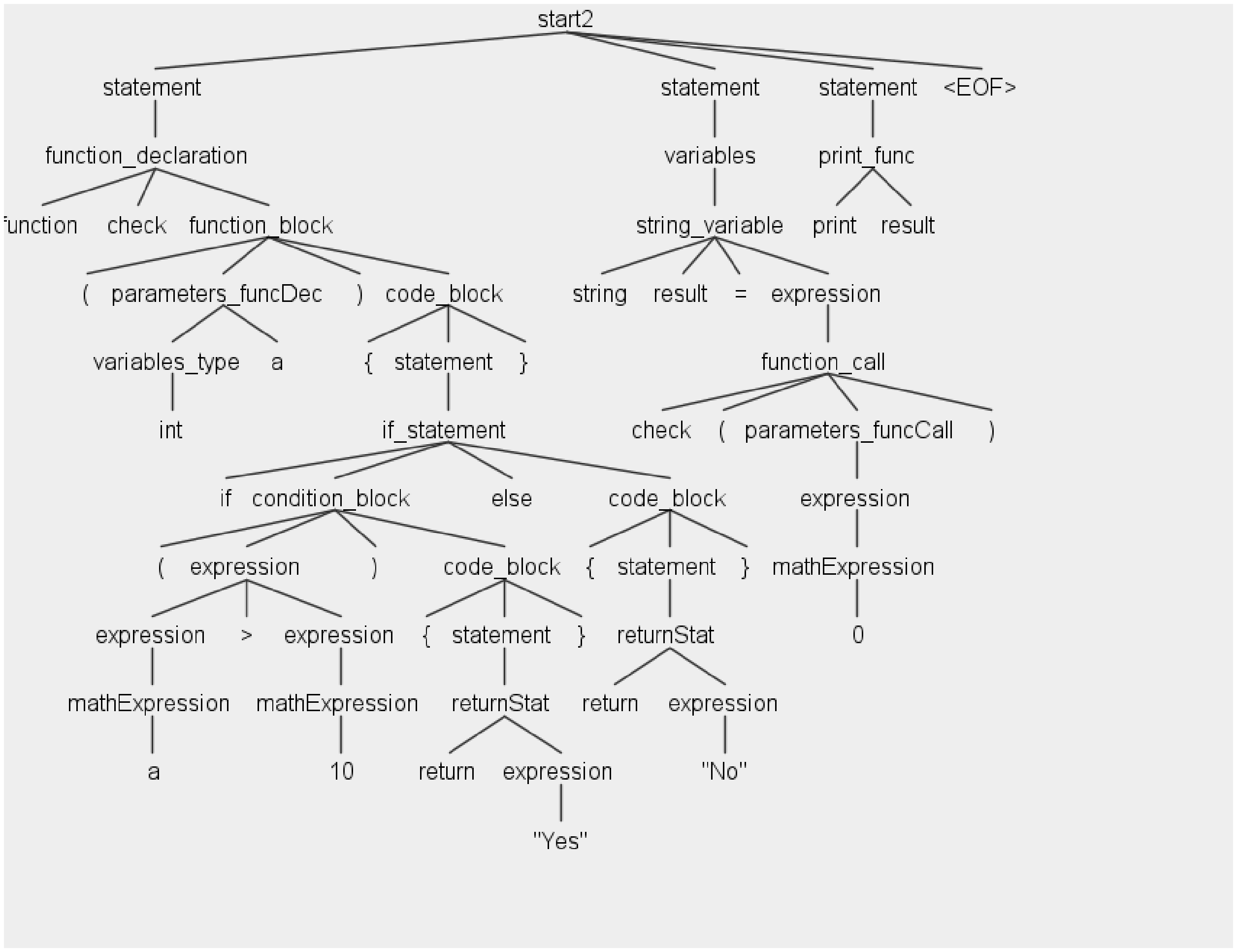
Terminal



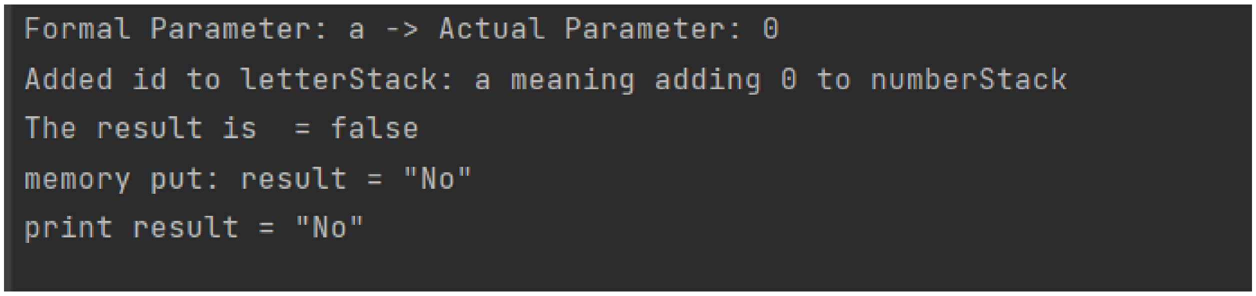
```
Formal Parameter: a -> Actual Parameter: 1
Formal Parameter: b -> Actual Parameter: 2
Added id to letterStack: a meaning adding 1 to numberStack
Added id to letterStack: b meaning adding 2 to numberStack
The result is  = false
Added id to letterStack: a meaning adding 1 to numberStack
Added id to letterStack: b meaning adding 2 to numberStack
memory put: c = 3
Formal Parameter: a -> Actual Parameter: 2
Added id to letterStack: c meaning adding 3 to numberStack
Added id to letterStack: d meaning adding -2 to numberStack
memory put: finalResult = -6
print c = 3
print d = -2
print finalResult = -6
```

```
6) function check(int a){
        if(a > 10){
           return "Yes"
        }
        else{
           return "No"
        }
   }
   string result = check(0)
   print result
```

Parse Tree



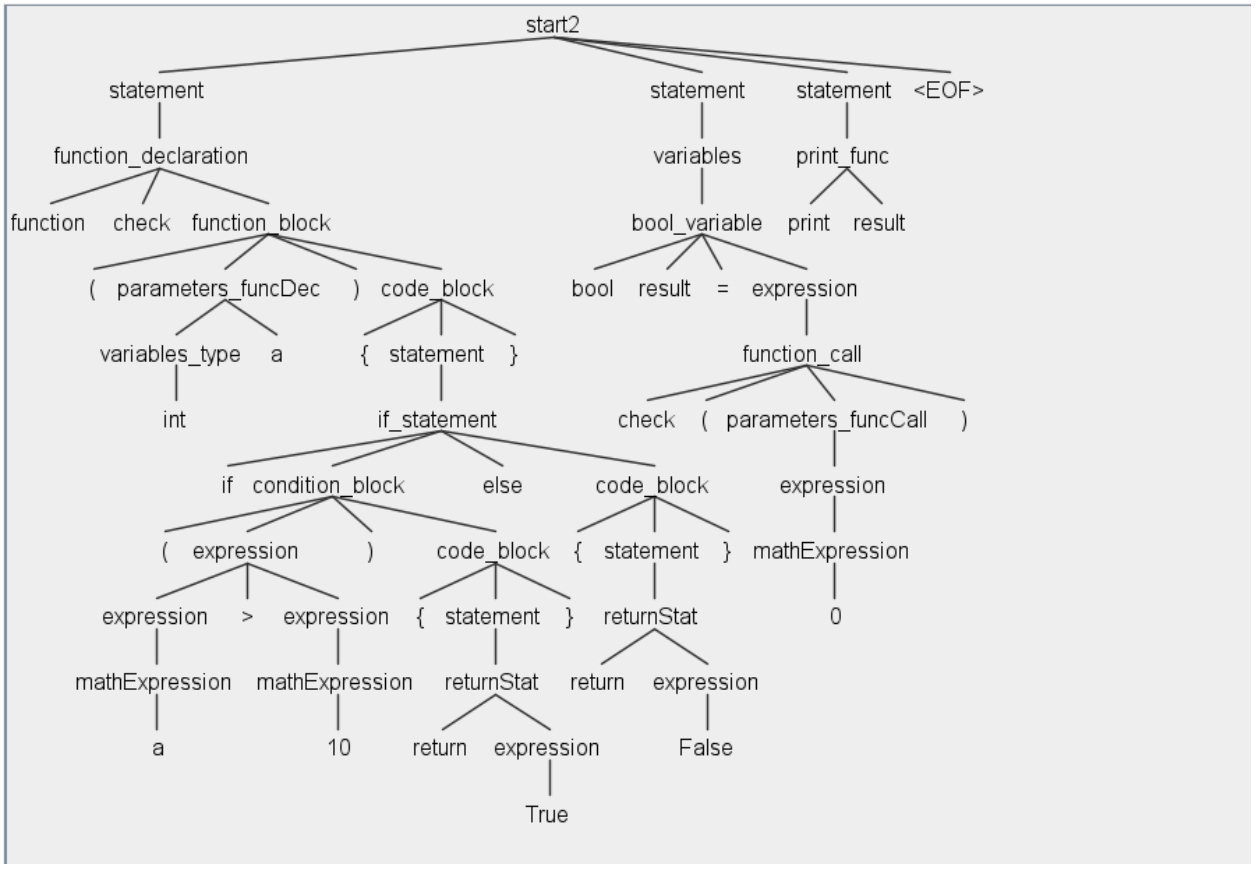
Terminal



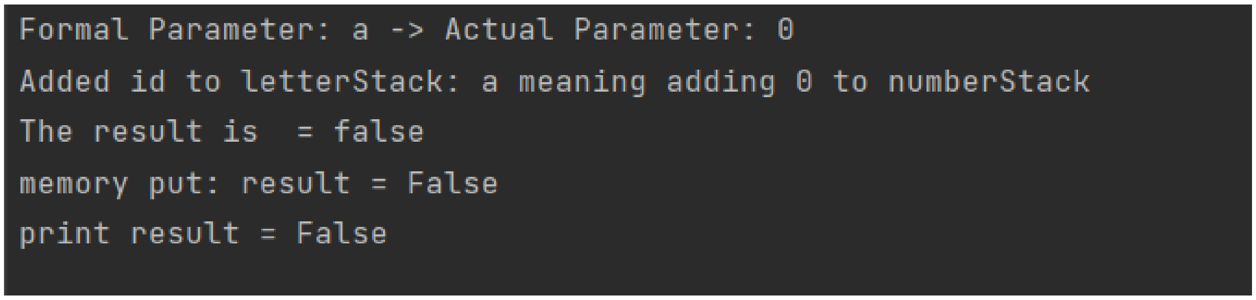
```
Formal Parameter: a -> Actual Parameter: 0
Added id to letterStack: a meaning adding 0 to numberStack
The result is  = false
memory put: result = "No"
print result = "No"
```

```
7) function check(int a){
        if(a > 10){
           return True
        }
        else{
           return False
        }
   }
   bool result = check(0)
   print result
```

Parse Tree



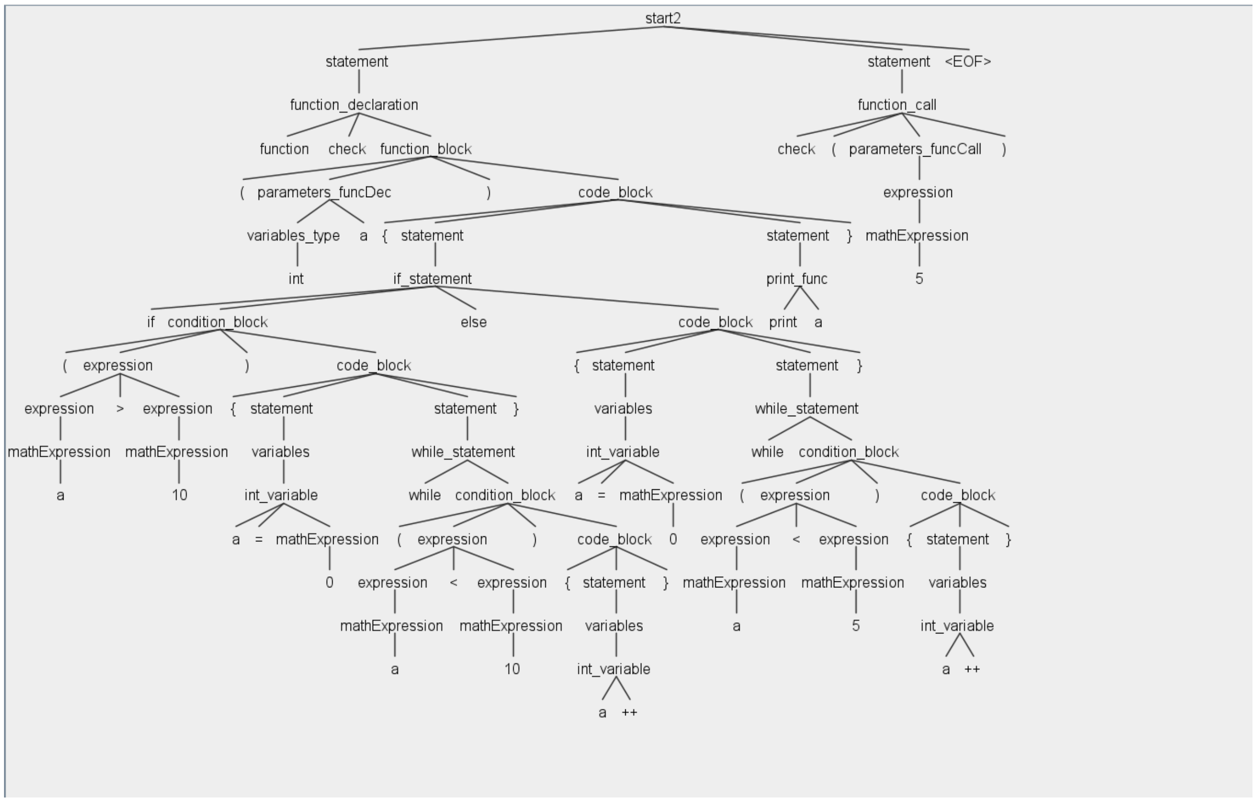
Terminal



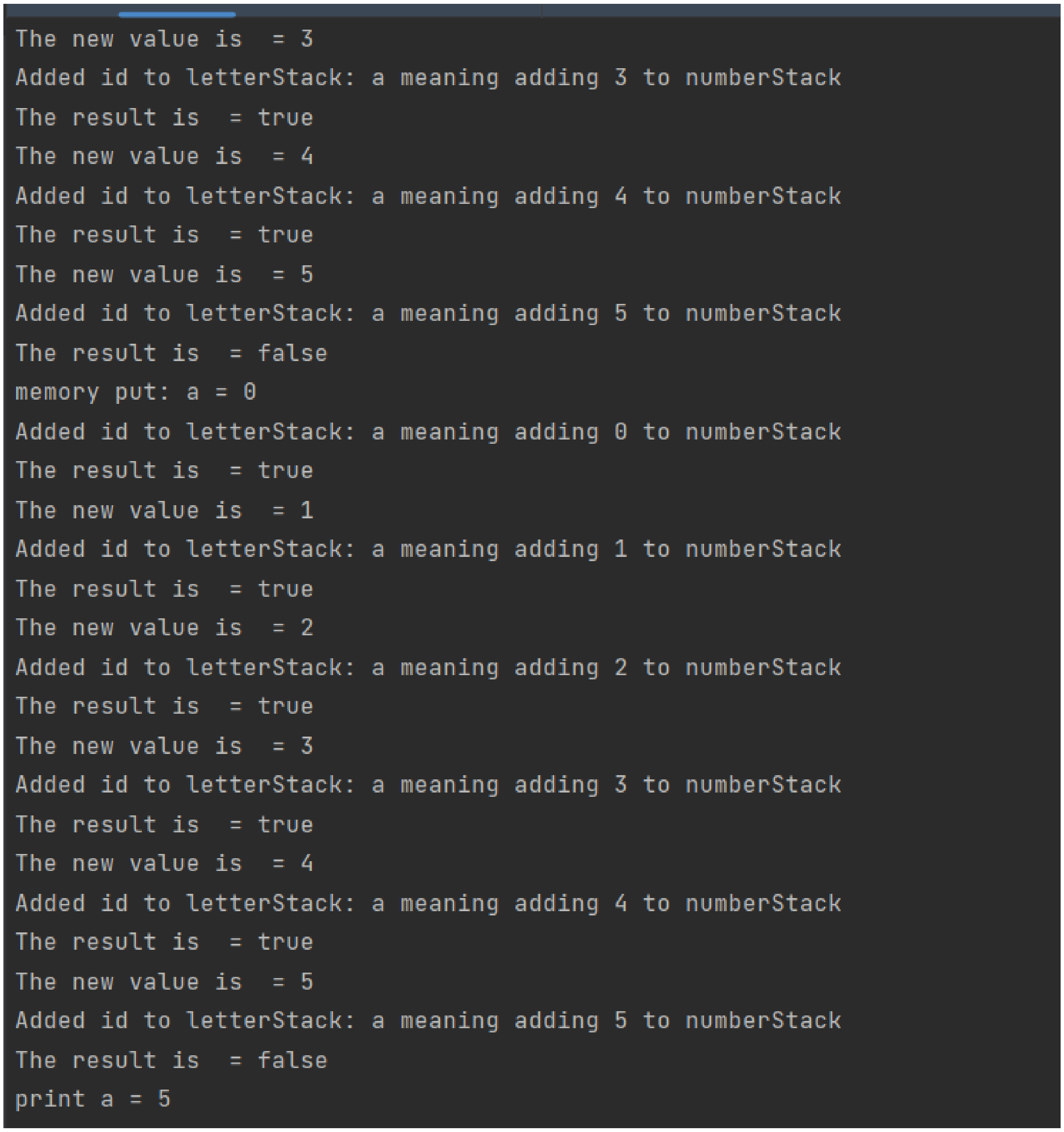
```
Formal Parameter: a -> Actual Parameter: 0
Added id to letterStack: a meaning adding 0 to numberStack
The result is  = false
memory put: result = False
print result = False
```

```
8) function check(int a){
       if(a > 10){
          a = 0
          while(a < 10){
              a++
          }
       }else{
          a = 0
          while(a < 5){
              a++
          }
       }
       print a
   }
   check(5)
```

Parse Tree



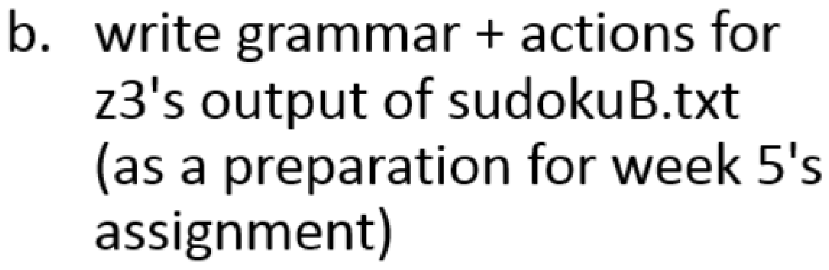
Terminal



```
The new value is  = 3
Added id to letterStack: a meaning adding 3 to numberStack
The result is  = true
The new value is  = 4
Added id to letterStack: a meaning adding 4 to numberStack
The result is  = true
The new value is  = 5
Added id to letterStack: a meaning adding 5 to numberStack
The result is  = false
memory put: a = 0
Added id to letterStack: a meaning adding 0 to numberStack
The result is  = true
The new value is  = 1
Added id to letterStack: a meaning adding 1 to numberStack
The result is  = true
The new value is  = 2
Added id to letterStack: a meaning adding 2 to numberStack
The result is  = true
The new value is  = 3
Added id to letterStack: a meaning adding 3 to numberStack
The result is  = true
The new value is  = 4
Added id to letterStack: a meaning adding 4 to numberStack
The result is  = true
The new value is  = 5
Added id to letterStack: a meaning adding 5 to numberStack
The result is  = false
print a = 5
```

Z3 SudokuB

Problem description


b. write grammar + actions for z3's output of sudokuB.txt (as a preparation for week 5's assignment)

Implementation

Since last week, we had to implement a new rule for ite (“If-then-else”) to parse the new z3 output.

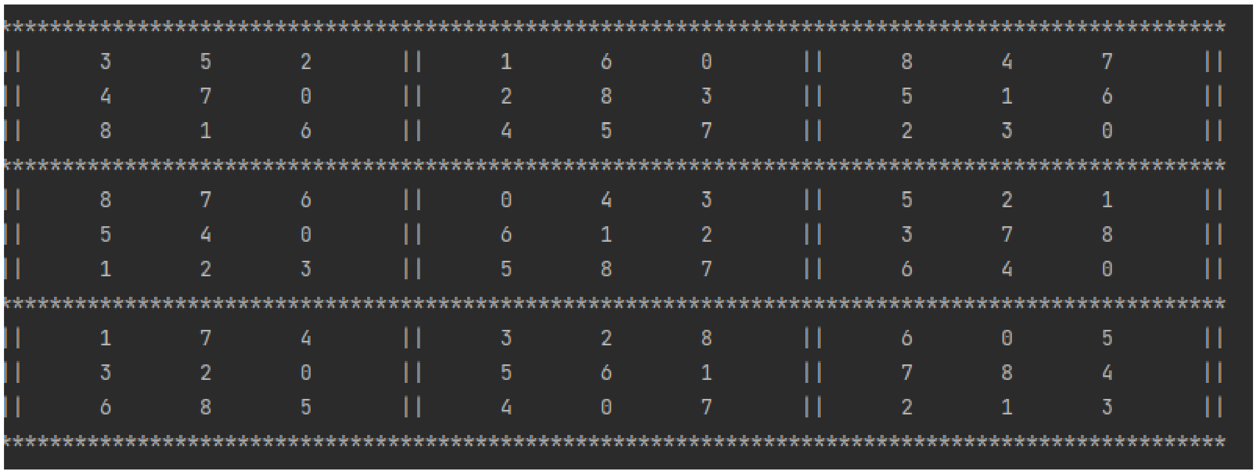
In order to add the results of the functions to the grid, when we visit the ite, we retrieve the ID’s of the variables used in the functions, and their the value so that, in the end, when we add it to the grid, we can specify which row/column should have the result value of the function.

The implementation can be found alongside this document.

```
grammar Example2;

// rules
start2    : statement* EOF;


statement : checkSatResponse
          | checkModelResponse
          ;

checkSatResponse    : SAT | UNSAT | UNKNOWN;

checkModelResponse  : PARANL model* PARANR;

model : (PARANL DEFINE_FUN ID PARANL global_declarations* PARANR INT) (NUMBER |
ite)  PARANR   #modelR;
```

```
ite: PARANL ITE PARANL AND get_assignment* PARANR NUMBER (NUMBER| ite)* PARANR;

get_assignment: PARANL EQUAL ID NUMBER PARANR;

global_declarations: PARANL ID INT PARANR;

//tokens
MODEL                  : 'model';
ITE                    : 'ite';
AND                    : 'and';
EQUAL                  :  '=';
INT                    : 'Int';
NUMBER                 : [0-9]+;
TEXT                   : '"' ~('\r' | '\n' | '"')* '"';
UNSAT                  : 'unsat';
SAT                    : 'sat';
DEFINE_FUN             : 'define-fun';
ID                     :  [_A-Za-z][A-Za-z_!0-9.]* ;
PARANL                 : '(';
PARANR                 : ')';
NEWLINE                : [ \t\r\n]+ -> skip;
```

Output



```
*************************************************************************************************
||      3       5       2       ||      1       6       0       ||      8       4       7       ||
||      4       7       0       ||      2       8       3       ||      5       1       6       ||
||      8       1       6       ||      4       5       7       ||      2       3       0       ||
*************************************************************************************************
||      8       7       6       ||      0       4       3       ||      5       2       1       ||
||      5       4       0       ||      6       1       2       ||      3       7       8       ||
||      1       2       3       ||      5       8       7       ||      6       4       0       ||
*************************************************************************************************
||      1       7       4       ||      3       2       8       ||      6       0       5       ||
||      3       2       0       ||      5       6       1       ||      7       8       4       ||
||      6       8       5       ||      4       0       7       ||      2       1       3       ||
*************************************************************************************************
```