Marshmallow SER 502 Team 15

Amaresh Bingumalla

Balachandar Sampath

Ejaz Saifudeen

Prasanth Venugopal

MARSHMALLOW

This is a Language that has been developed from scratch using C++ and STL

The language has been developed based on the Imperative programming paradigm.

The grammar of the language was made to follow the recursive descent model

The language can support Assignment, Arithmetic operations, Logical operations, Comparison operators, Assignment operations, Identifiers, Print statements, Continue statements, Return statements, Break statements, Conditional statements, Loops statements , Functions

Grammar

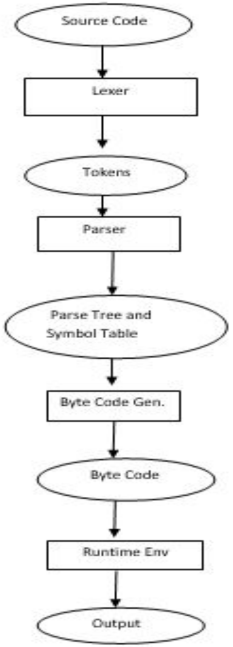
```
marshmallow ::= (stmt)+
letter ::= [a-zA-Z]
digit ::= [0-9]
identifier ::= (letter|"_") (letter | digit | "_")*
funcname ::= identifier
integer ::= digit(digit)*
stmt   ::= assignment_stmt NEWLINE
          | print_stmt NEWLINE
          | return_stmt NEWLINE
          | break_stmt NEWLINE
          | continue_stmt NEWLINE
          | exec_stmt NEWLINE
       | if_stmt
          | while_stmt
          | func_def
```

Grammar Contd..

```
block ::= NEWLINE INDENT (stmt )+ DEDENT
if_stmt ::= "if" expression block
        ( "elif" expression block )*
        ("else" block)?
while_stmt ::= "while" expression block
funcdef ::= "function" funcname "(" parameters ")" block
parameters ::= identifier (',' identifier)*
print ::= “print” expression
assignment_stmt ::=
identifier::= "=" expression
return ::= “return” expression
break ::= “break”
continue ::= “continue”
arguments ::= expression (‘,’expression)*
exec_stmt ::= funcname "(" arguments ")"
```

Design flow

1. Source Code
2. Lexer
3. Tokens
4. Parser
5. Parse Tree and Symbol Table
6. ByteCode Generation
7. Byte Code
8. Runtime Env
9. Output



Grammar Contd..

```
expression ::= or_expr
or_expr ::= and_expr | and_expr "or" or_expr
and_expr ::= not_expr | not_expr "and" and_expr
not_expr ::= comp_expr | "not" not_expr
comp_expr ::= a_expr | a_expr  comp_opr comp_expr
a_expr ::= m_expr | m_expr "+" a_expr | m_expr "-" a_expr
m_expr ::= unit| unit "*" m_expr | unit "/" m_expr | unit "%" m_expr
unit ::= integer | identifier | “(” expression“)” | exec_stmt
comp_opr ::=
        "<" | ">" | "==" | ">=" | "<=" | "<>"
```

Steps involved in creating language

1. Lexer: This takes in the program as input and creates tokens out of the program.
   i. This analyzes the input program to create tokens.
2. Parser: Tokens from the Lexer is fed as input to Parser which generates Parse Tree and Symbol table.
   i. This is a recursive descent parser with look ahead.
   ii. This part takes care of semantic analysis and generating the parse tree.
   iii. This generates symbol table which is a doubly linked N-ary tree structure.
   iv. This throws an error if the given program has syntax errors.
3. Intermediate Code: This generates the bytecode based on the parse tree in which is in agreement with the runtime.
   i. This traverses the parse tree and generates the bytecode using opcodes.
4. Runtime Environment: This takes the bytecode and does the execution of the program written. This gives out the output after completing the execution.
   i. Stack Model is used for execution.
   ii. Bytecode is traversed and executed with the stack holding current values.

SAMPLE PROGRAM EXECUTION

```
a = 5
i = 1
if (a%2 == 0)
print (a)
else
while (i<a)
print(i)
i = i+1
```

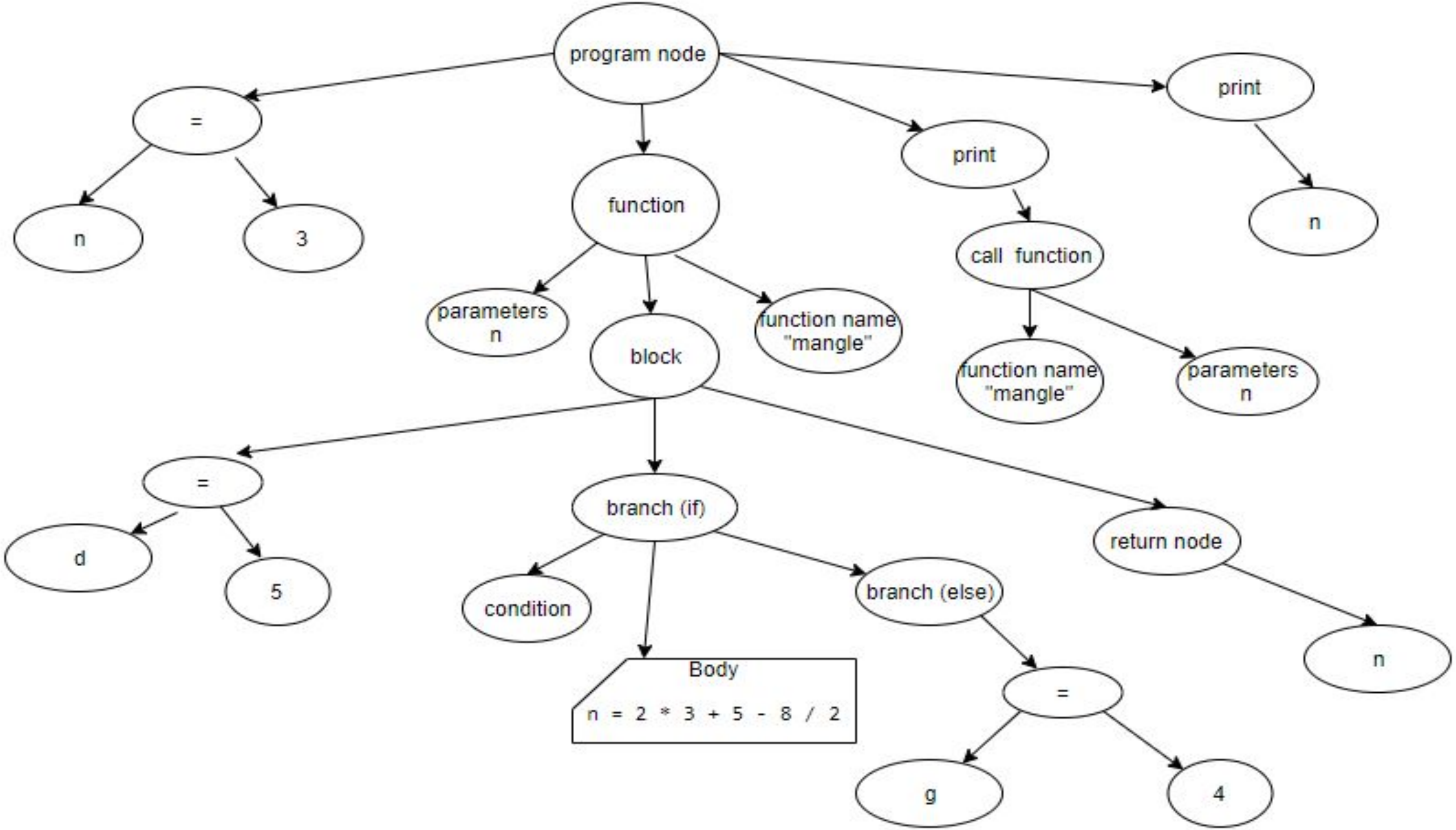
LEXER

```
String[] program = [“a”,”=”,”5”,”NEWLINE”,”i’
,”=”,”1”,”NEWLINE","if”,"(","a","%","2","==","0",")","NEWLINE","INDENT","print","(",
"a",")","NEWLINE","DEDENT","else,"NEWLINE","INDENT","while","(","i","<","a",")"
,",NEWLINE","INDENT","print","(","i",")","NEWLINE","i","=","i","+”,"1","NEWLINE","
DEDENT", "DEDENT"};
```

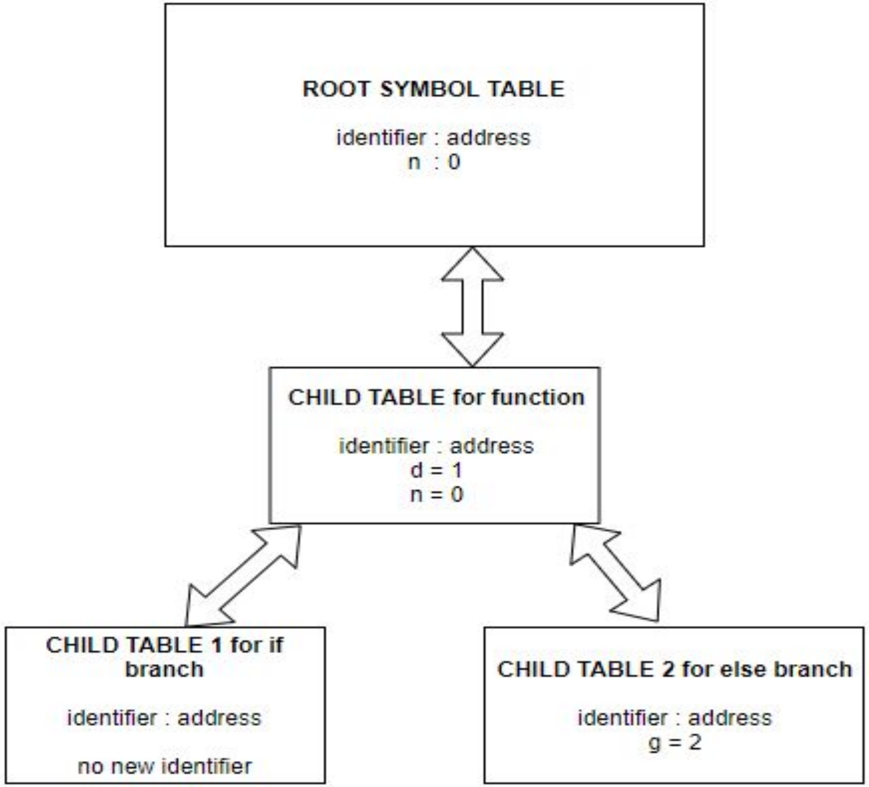
SAMPLE PROGRAM

```
n = 3
function mangle(n)
    d = 5
    if 2 < 3
        n = 2 * 3 + 5 - 8 / 2
    else
        g = 4
return n
print mangle(n)
print n
```

Structure of Parse Tree



SYMBOL TABLE



ROOT SYMBOL TABLE

identifier : address
n : 0

CHILD TABLE for function

identifier : address
d = 1
n = 0

CHILD TABLE 1 for if branch

identifier : address

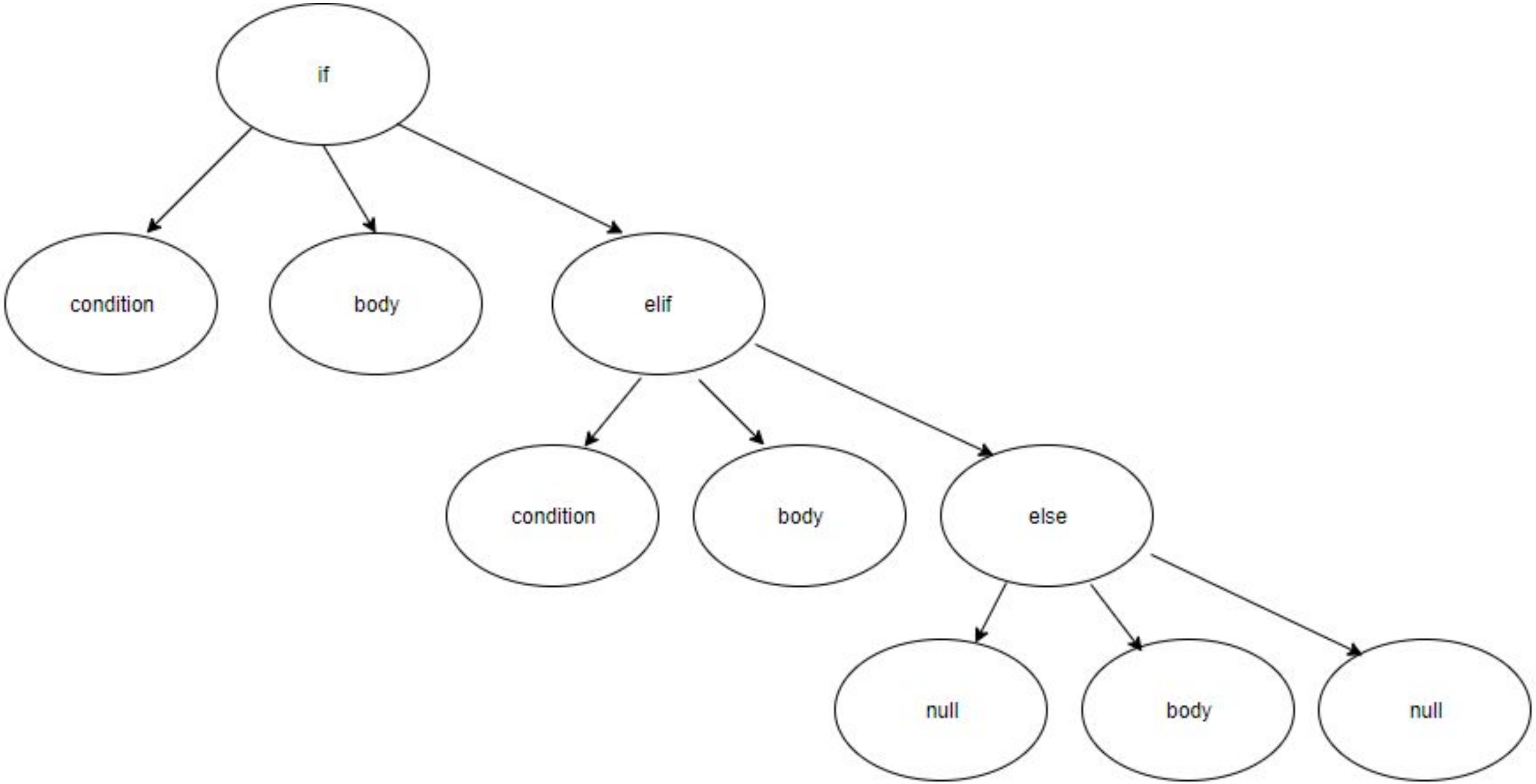
no new identifier

CHILD TABLE 2 for else branch

identifier : address
g = 2

ByteCode

```
0 PUSH 5
2 STORE #0
4 PUSH 1
6 STORE #1
8 PUSH 0
10 PUSH 2
12 LOAD #0
14 MOD
15 EQ
16 BRF $22
18 LOAD #0
20 PRINT
21 BR $40
23 PUSH 10
25 LOAD #1
27 LT
28 BRF $40
30 LOAD #1
32 PRINT
33 PUSH 1
35 LOAD #1
37 ADD
38 STORE #1
40 EXIT
```

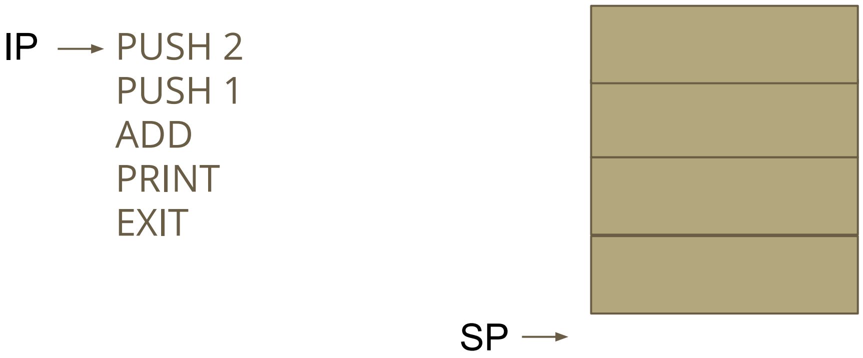

if
condition
body
elif
condition
body
else
null
body
null

Runtime

```
PUSH 2
PUSH 1
ADD
PRINT
EXIT
```

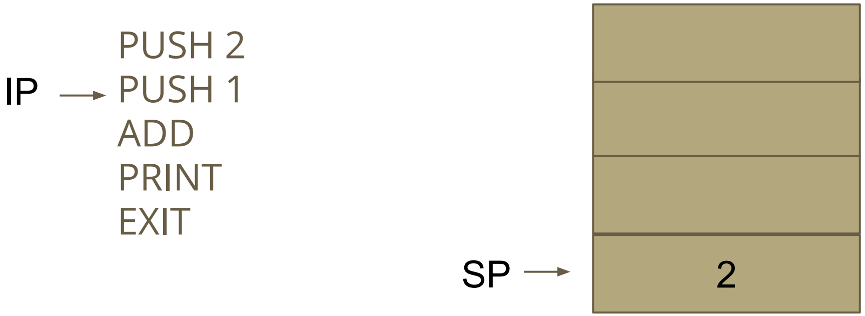
```
print 1 + 2
```

Runtime

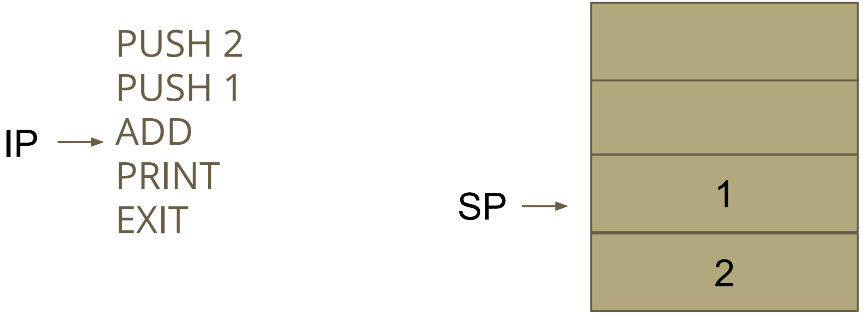


```
PUSH 2
PUSH 1
ADD
PRINT
EXIT
```

Runtime



Runtime



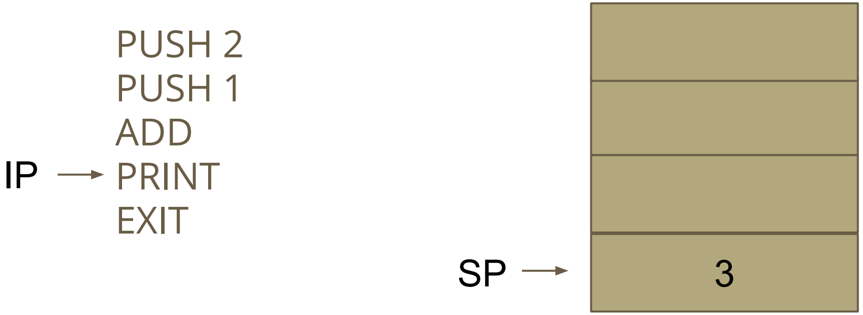
```
      PUSH 2
      PUSH 1
IP -> ADD
      PRINT
      EXIT
```

SP

1

2

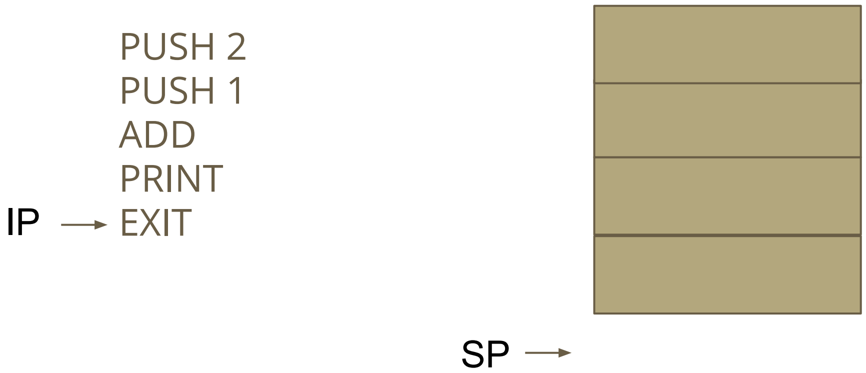
Runtime



PUSH 2
PUSH 1
ADD
IP → PRINT
EXIT

SP → 3

Runtime



```
PUSH 2
PUSH 1
ADD
PRINT
EXIT
```

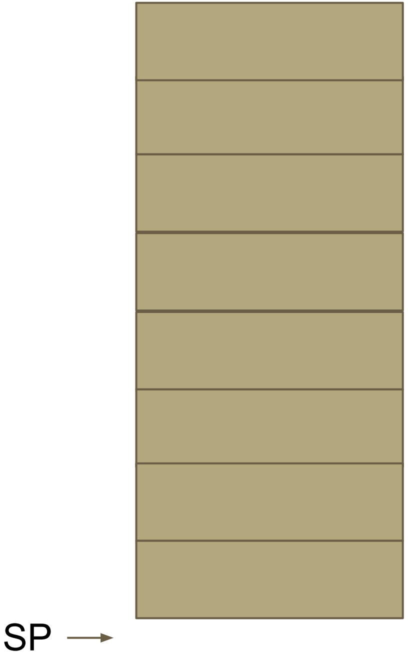
Runtime

```
0  BR 25
2  PUSH 1
4  LOAD 0
6  LTE
8  BRF 13
10  PUSH 1
12  RET
13  PUSH 1
15  LOAD 0
17  SUB
18  CALL 2 1
21  LOAD 0
23  MULT
24  RET
25  PUSH 3
27  CALL 2 1
30  PRINT
```

```
Function fact(n)
    if(n <= 1)
        return 1
    return(n *fact (n-1))
fact(3)
```

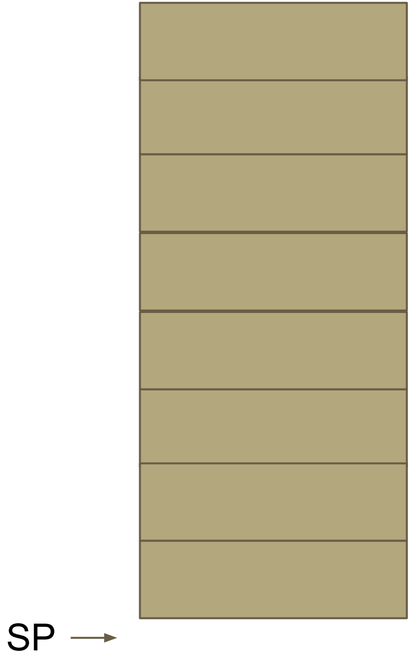
Runtime

```
IP →  0  BR 25
      2  PUSH 1
      4  LOAD 0
      6  LTE
      8  BRF 13
      10  PUSH 1
      12  RET
      13  PUSH 1
      15  LOAD 0
      17  SUB
      18  CALL 2 1
      21  LOAD 0
      23  MULT
      24  RET
      25  PUSH 3
      27  CALL 2 1
      30  PRINT
```



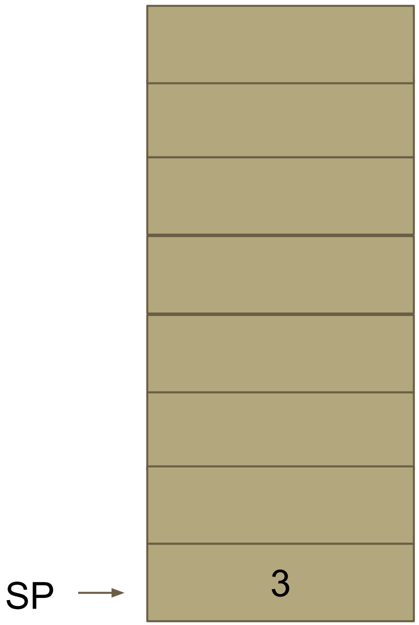
Runtime

```
      0  BR 25
      2  PUSH 1
      4  LOAD 0
      6  LTE
      8  BRF 13
      10  PUSH 1
      12  RET
      13  PUSH 1
      15  LOAD 0
      17  SUB
      18  CALL 2 1
      21  LOAD 0
      23  MULT
      24  RET
IP →  25  PUSH 3
      27  CALL 2 1
      30  PRINT
```

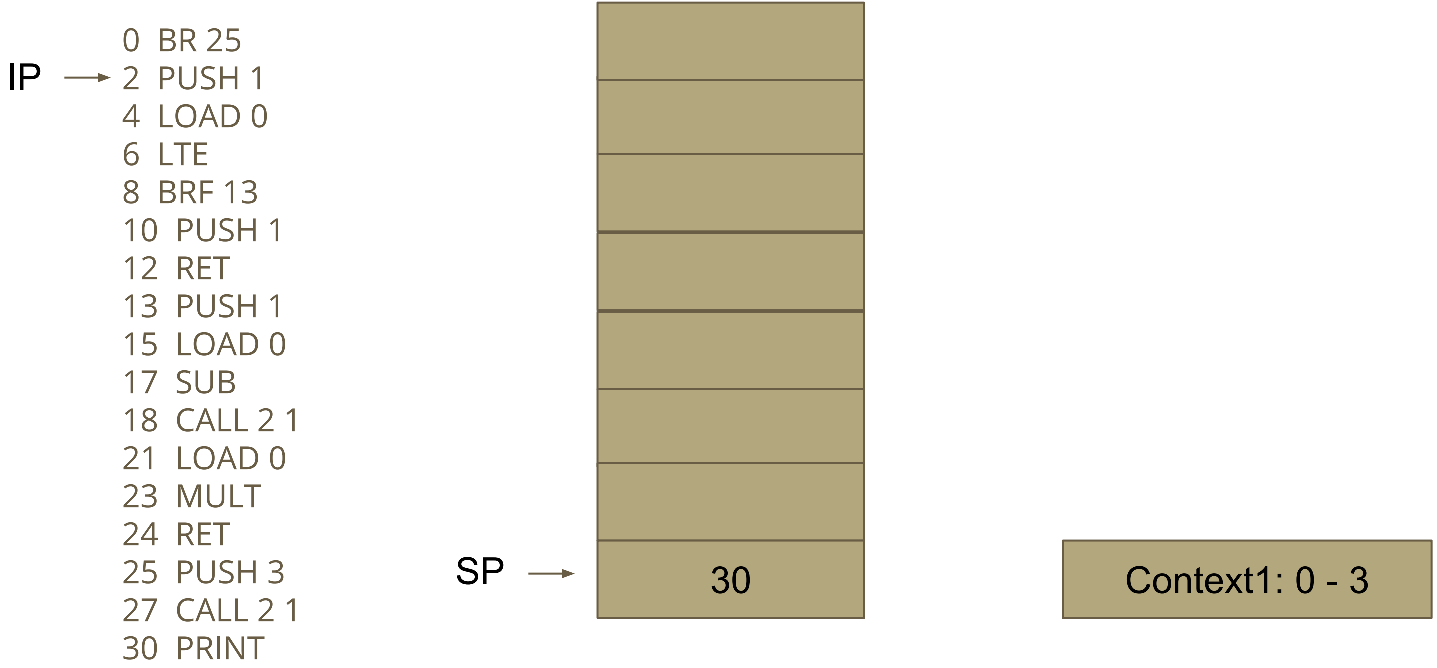


Runtime

```
        0  BR 25
        2  PUSH 1
        4  LOAD 0
        6  LTE
        8  BRF 13
        10  PUSH 1
        12  RET
        13  PUSH 1
        15  LOAD 0
        17  SUB
        18  CALL 2 1
        21  LOAD 0
        23  MULT
        24  RET
        25  PUSH 3
IP  →   27  CALL 2 1
        30  PRINT
```



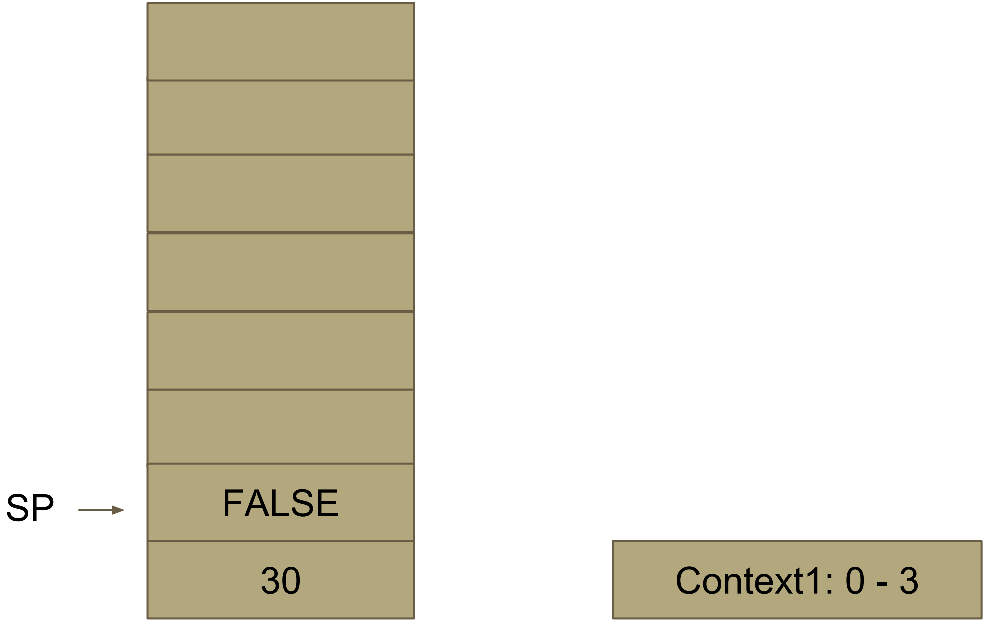
Runtime



```
0  BR 25
2  PUSH 1
4  LOAD 0
6  LTE
8  BRF 13
10  PUSH 1
12  RET
13  PUSH 1
15  LOAD 0
17  SUB
18  CALL 2 1
21  LOAD 0
23  MULT
24  RET
25  PUSH 3
27  CALL 2 1
30  PRINT
```

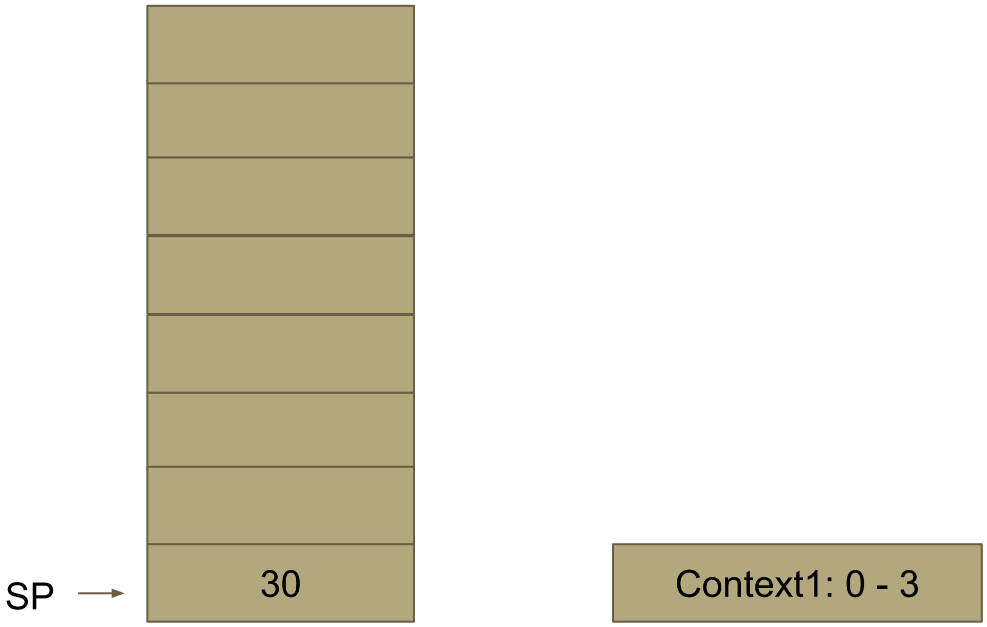
Runtime

```
      0  BR 25
      2  PUSH 1
      4  LOAD 0
      6  LTE
IP →  8  BRF 13
      10  PUSH 1
      12  RET
      13  PUSH 1
      15  LOAD 0
      17  SUB
      18  CALL 2 1
      21  LOAD 0
      23  MULT
      24  RET
      25  PUSH 3
      27  CALL 2 1
      30  PRINT
```



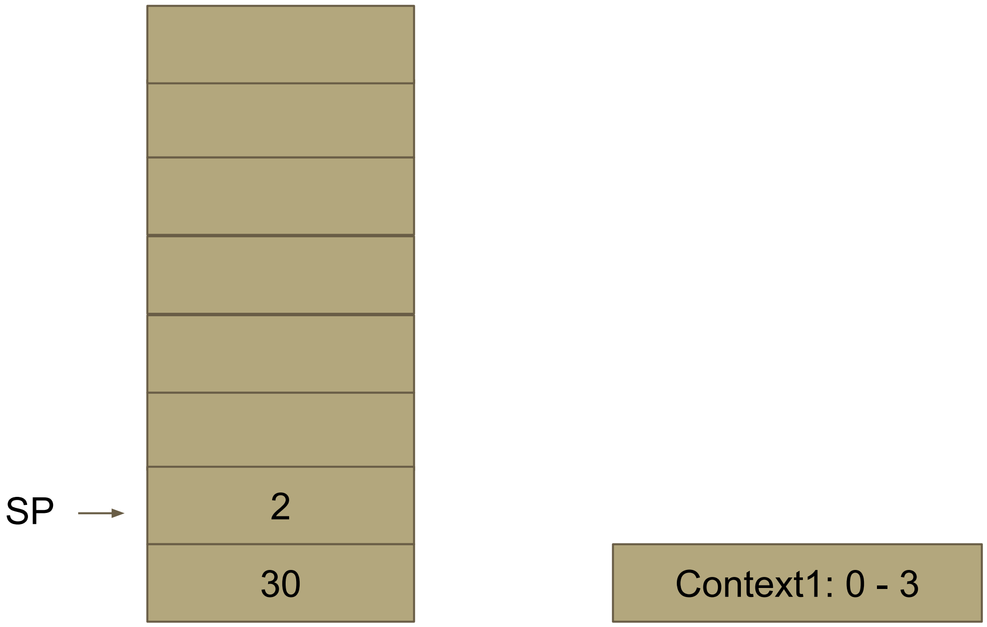
Runtime

```
      0  BR 25
      2  PUSH 1
      4  LOAD 0
      6  LTE
      8  BRF 13
      10  PUSH 1
      12  RET
IP →  13  PUSH 1
      15  LOAD 0
      17  SUB
      18  CALL 2 1
      21  LOAD 0
      23  MULT
      24  RET
      25  PUSH 3
      27  CALL 2 1
      30  PRINT
```

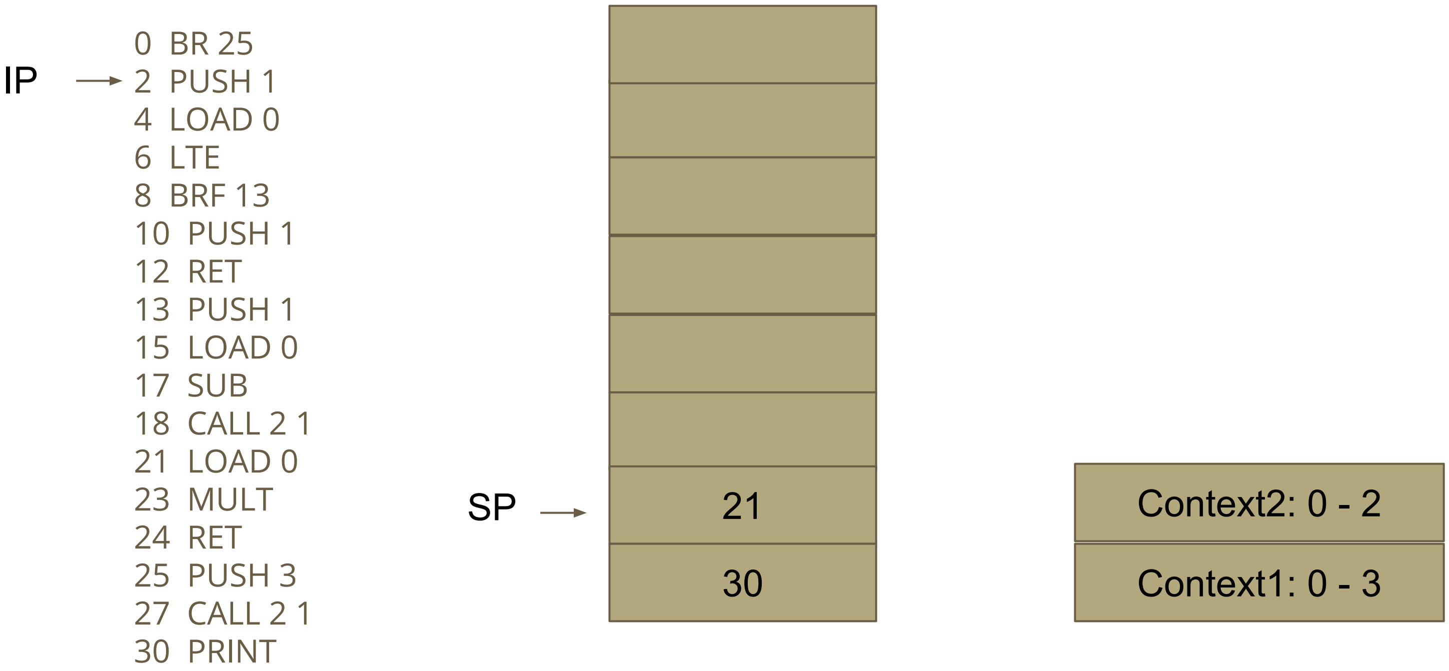


Runtime

```
     0  BR 25
     2  PUSH 1
     4  LOAD 0
     6  LTE
     8  BRF 13
     10  PUSH 1
     12  RET
     13  PUSH 1
     15  LOAD 0
     17  SUB
IP → 18  CALL 2 1
     21  LOAD 0
     23  MULT
     24  RET
     25  PUSH 3
     27  CALL 2 1
     30  PRINT
```



Runtime



```
IP  →  0  BR 25
       2  PUSH 1
       4  LOAD 0
       6  LTE
       8  BRF 13
       10  PUSH 1
       12  RET
       13  PUSH 1
       15  LOAD 0
       17  SUB
       18  CALL 2 1
       21  LOAD 0
       23  MULT
       24  RET
       25  PUSH 3
       27  CALL 2 1
       30  PRINT
```

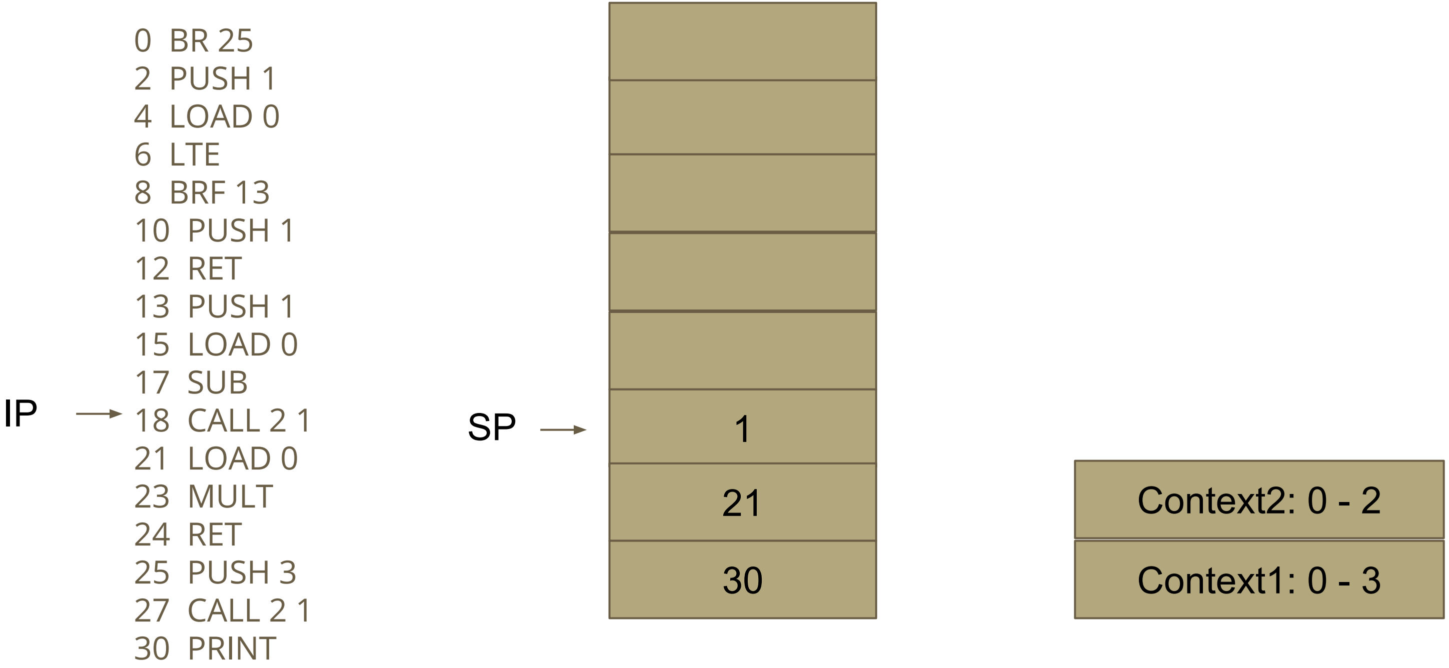
SP → 21

30

Context2: 0 - 2

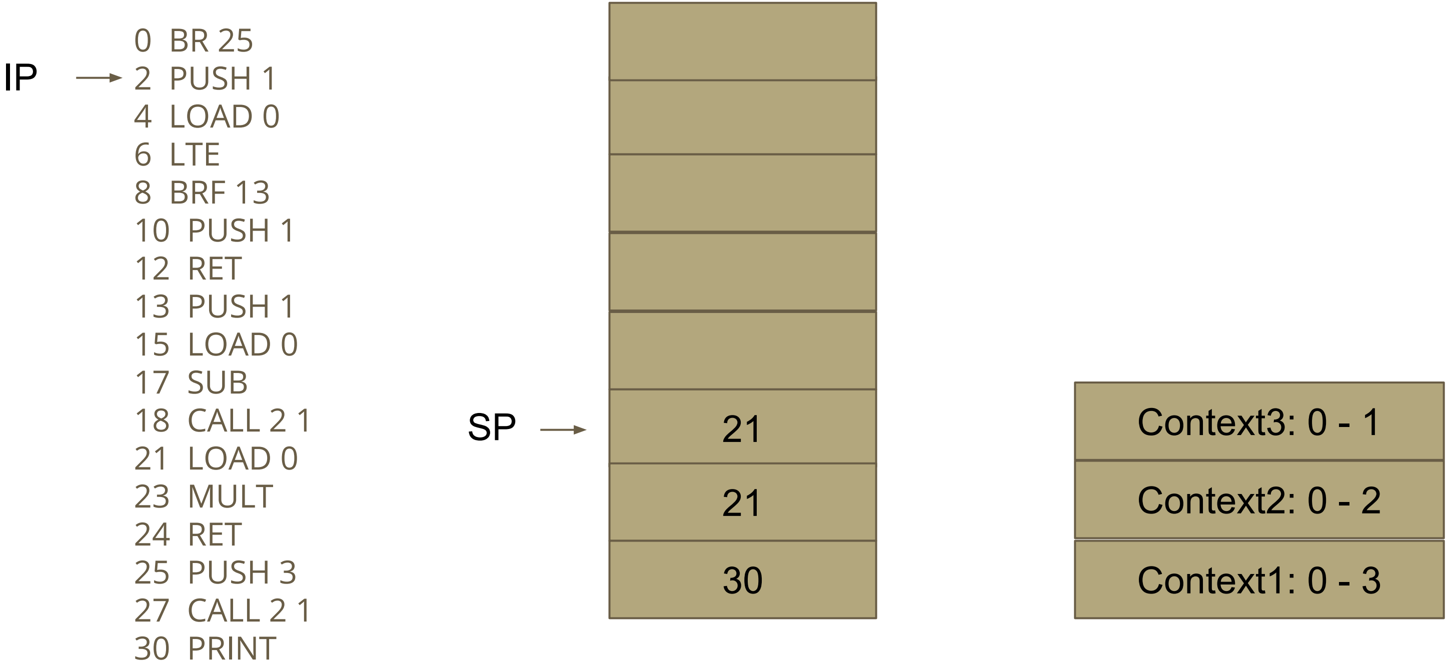
Context1: 0 - 3

Runtime



```
0  BR 25
2  PUSH 1
4  LOAD 0
6  LTE
8  BRF 13
10  PUSH 1
12  RET
13  PUSH 1
15  LOAD 0
17  SUB
18  CALL 2 1
21  LOAD 0
23  MULT
24  RET
25  PUSH 3
27  CALL 2 1
30  PRINT
```

Runtime



IP →

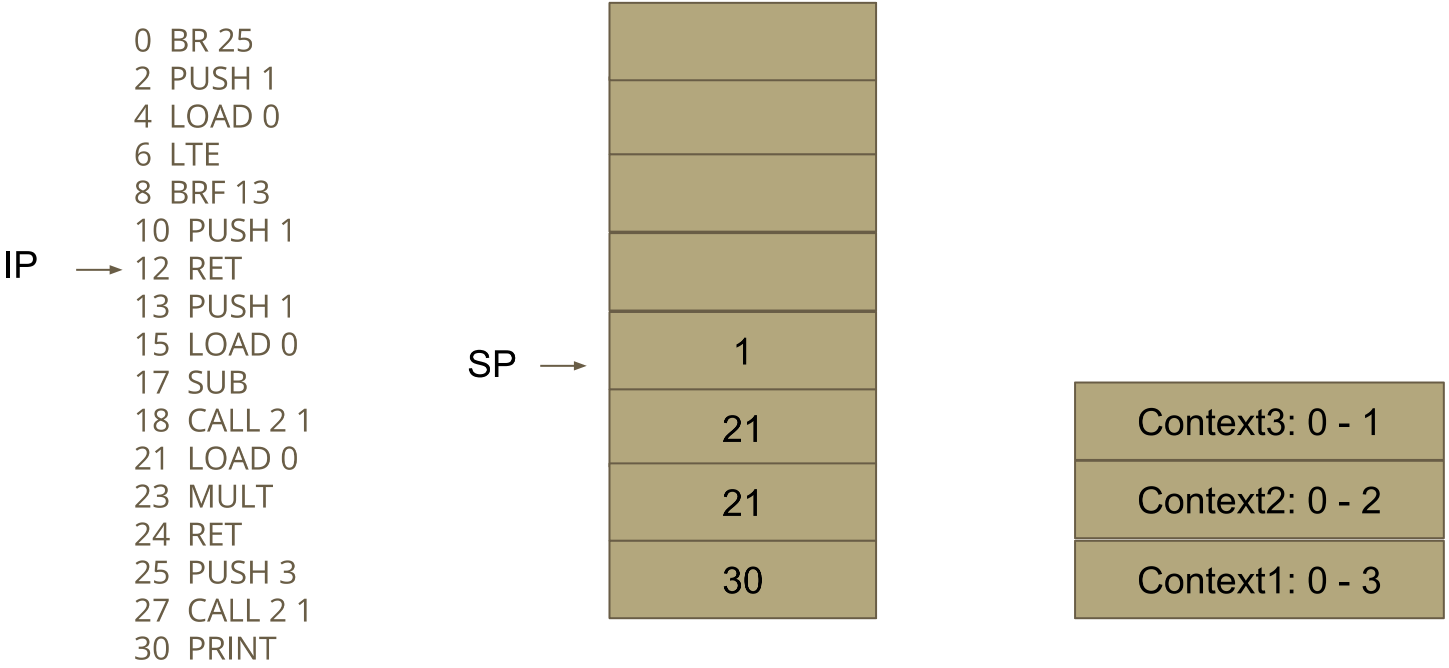
```
0  BR 25
2  PUSH 1
4  LOAD 0
6  LTE
8  BRF 13
10  PUSH 1
12  RET
13  PUSH 1
15  LOAD 0
17  SUB
18  CALL 2 1
21  LOAD 0
23  MULT
24  RET
25  PUSH 3
27  CALL 2 1
30  PRINT
```

SP →

21
21
30

Context3: 0 - 1
Context2: 0 - 2
Context1: 0 - 3

Runtime



```
0  BR 25
2  PUSH 1
4  LOAD 0
6  LTE
8  BRF 13
10  PUSH 1
12  RET
13  PUSH 1
15  LOAD 0
17  SUB
18  CALL 2 1
21  LOAD 0
23  MULT
24  RET
25  PUSH 3
27  CALL 2 1
30  PRINT
```

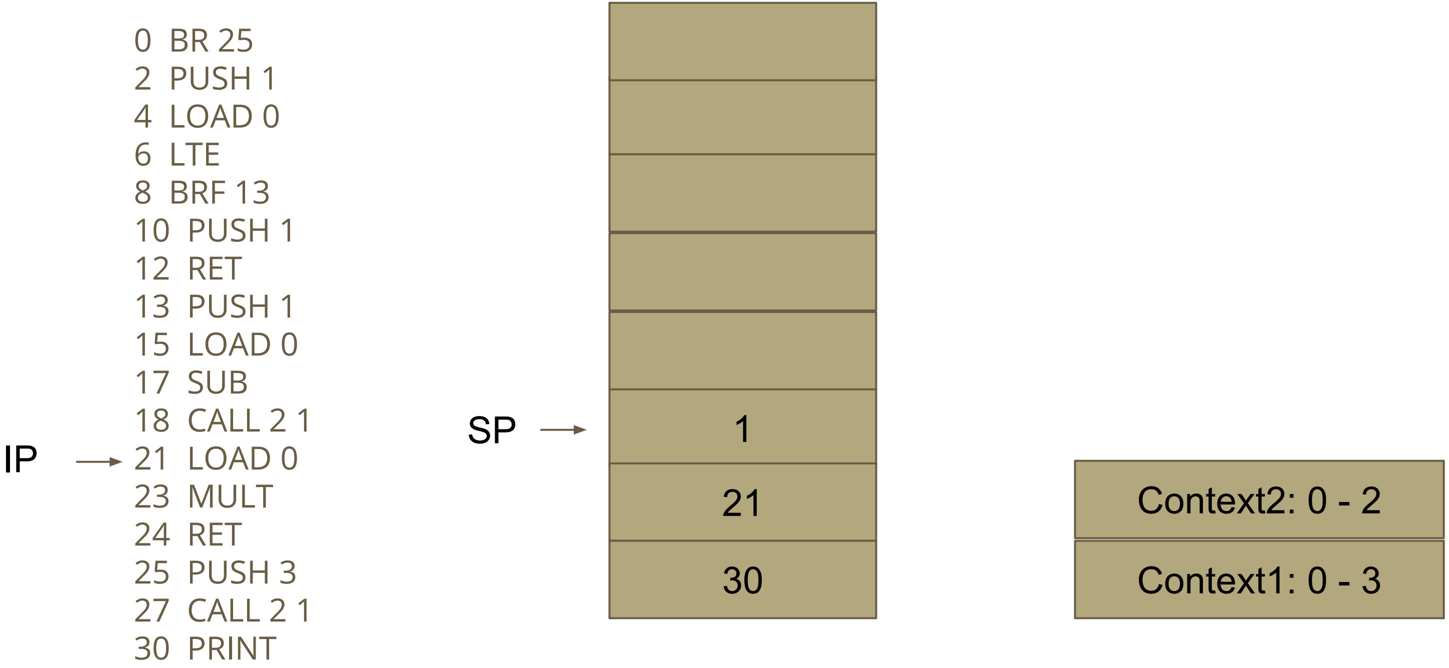
IP → 12 RET

SP → 1

1
21
21
30

Context3: 0 - 1
Context2: 0 - 2
Context1: 0 - 3

Runtime



```
      0  BR 25
      2  PUSH 1
      4  LOAD 0
      6  LTE
      8  BRF 13
      10  PUSH 1
      12  RET
      13  PUSH 1
      15  LOAD 0
      17  SUB
      18  CALL 2 1
IP →  21  LOAD 0
      23  MULT
      24  RET
      25  PUSH 3
      27  CALL 2 1
      30  PRINT
```

SP → 1

21

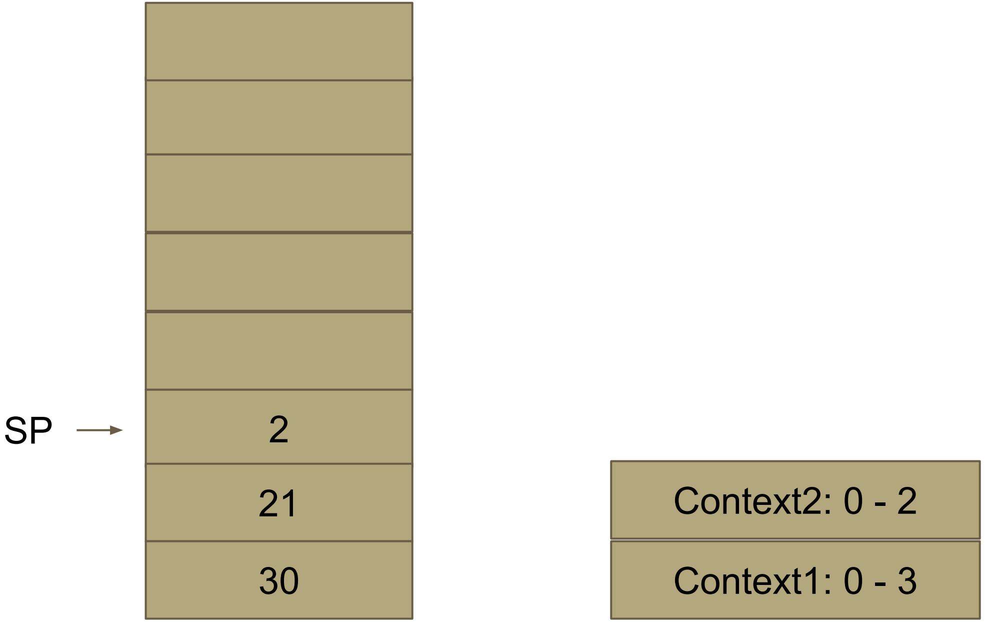
30

Context2: 0 - 2

Context1: 0 - 3

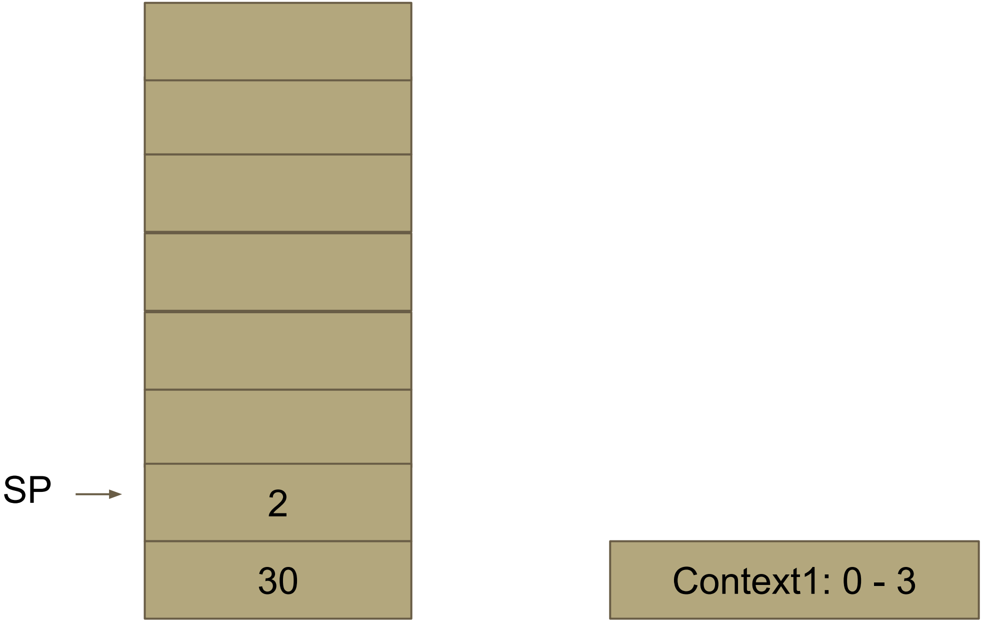
Runtime

```
          0  BR 25
          2  PUSH 1
          4  LOAD 0
          6  LTE
          8  BRF 13
          10  PUSH 1
          12  RET
          13  PUSH 1
          15  LOAD 0
          17  SUB
          18  CALL 2 1
          21  LOAD 0
          23  MULT
IP   →    24  RET
          25  PUSH 3
          27  CALL 2 1
          30  PRINT
```



Runtime

```
      0  BR 25
      2  PUSH 1
      4  LOAD 0
      6  LTE
      8  BRF 13
      10  PUSH 1
      12  RET
      13  PUSH 1
      15  LOAD 0
      17  SUB
      18  CALL 2 1
IP →  21  LOAD 0
      23  MULT
      24  RET
      25  PUSH 3
      27  CALL 2 1
      30  PRINT
```



SP →

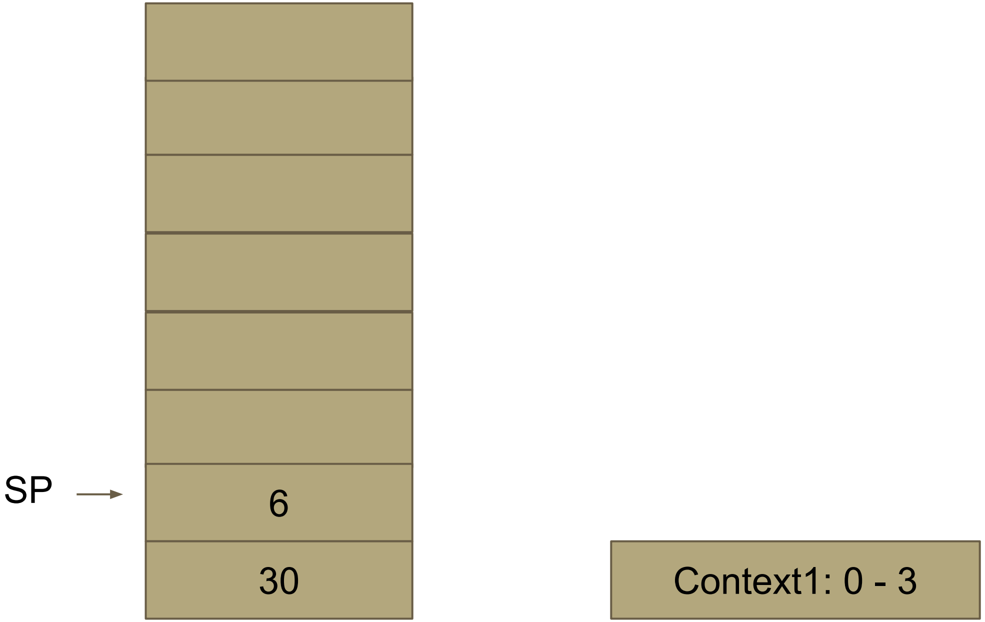
2

30

Context1: 0 - 3

Runtime

```
        0  BR 25
        2  PUSH 1
        4  LOAD 0
        6  LTE
        8  BRF 13
        10  PUSH 1
        12  RET
        13  PUSH 1
        15  LOAD 0
        17  SUB
        18  CALL 2 1
        21  LOAD 0
        23  MULT
IP  →   24  RET
        25  PUSH 3
        27  CALL 2 1
        30  PRINT
```



Runtime

```
      0  BR 25
      2  PUSH 1
      4  LOAD 0
      6  LTE
      8  BRF 13
      10  PUSH 1
      12  RET
      13  PUSH 1
      15  LOAD 0
      17  SUB
      18  CALL 2 1
      21  LOAD 0
      23  MULT
      24  RET
      25  PUSH 3
      27  CALL 2 1
IP →  30  PRINT
```