```
var x : int           #Global variable x
var y : bool     #Global variable y
function f()     #function f has no input parameter
     function g()     #function g nested in function f and has no input
          var x : int #local variable of function g
          begin       #body of the function g
               write x
          end
     begin            #body of the function f
          g()         #f calls g
          read x
          write x
          write y
     end
begin
     f()              #main function calls f
end
```

0 and 1: When we see a variable, we just emit a code that creates a space for the variable. We have two variables so we emit two lines of code to create space for them first.
2: We have a function definition, so we need to skip it and jump over the code regarding the function since we want to start running from main function.br 38 means update next = ip + 38 = 2 + 38 = 40
Note: We don’t know where to jump initially. We will update them after generating below code.

Note: Prologue-You push enough information to the stack so that you can restore the state before calling
a function.

Prologue (before emitting code for the block) for f()
3: f()frame return address
4: f()frame dynamic link
5: f()frame update the frame pointer to be the current stack pointer
Prologue for g()
7: g()frame return address
8: g()frame dynamic link
9: g()frame update the frame pointer to be the current stack pointer
10: local variable x in g()
block of function g()
11: It does not have any nested function declarations, hence, br 1
12: load local variable x
13: write local variable x

Epilogue for function g()
14: Tear down the stack frame
15: Update to the old frame pointer
16: Get the old instruction pointer (get return address)
17: g()return link register

18: g() frame return value
19: g() frame static link

21: move sp to g() return value
22: get g() return value

23: Read in stdin
24: save the static link (callee is defined inside the current scope)
25 and 26: copy the static link from the current stack frame (callee is a sibling)
27: Restore the current frame pointer

#write x
28: get the value of the frame pointer
29: the difference in nesting depth is 1. load static parent frame pointer once.
30: read the value of the global variable x to register 0
31: write the value of global var x, which lies in register 0

#write y
32: get the value of the frame pointer
33: the difference in nesting depth is 1. load static parent frame pointer once.
34: read the value of the global variable y to register 0
35: write the value of global var y, which lies in register 0

Epilogue for function f()
36: tear down the stack frame
37: update to the old frame pointer
38: get the old instruction pointer (get return address)

39: f() return link register
40: f() frame return value
41: f() frame static link
43: move sp to f() return value
44: get f() return value

45: hlt

```
0    addi sp sp 1
1    addi sp sp 1
2    br 38
3    psh ln sp
4    psh fp sp
5    mov fp sp
6    br 12
7    psh ln sp
8    psh fp sp
9    mov fp sp
10   addi sp sp 1
11   br 1
12   ld r0 fp 1
13   wr r0
14   mov sp fp
15   pop fp sp
16   pop ln sp
17   ret ln
18   addi sp sp 1
19   psh fp sp
20   bl -13
21   subi sp sp 1
22   pop r0 sp
23   read r0
24   psh fp sp
25   ld fp fp -2
26   st r0 fp 1
27   pop fp sp
28   mov r0 fp
29   ld r0 r0 -2
30   ld r0 r0 1
31   wr r0
32   mov r0 fp
33   ld r0 r0 -2
34   ld r0 r0 2
35   wr r0
36   mov sp fp
37   pop fp sp
38   pop ln sp
39   ret ln
40   addi sp sp 1
41   psh fp sp
42   bl -39
43   subi sp sp 1
44   pop r0 sp
45   hlt
```