Consider a language of concatenated instructions
The instructions can be blocks or other instructions.
A block consists of a declaration of variables (type: int or real)
followed by a concatenation of instructions.

Define an SDT suitable for TOP-DOWN parsing that calculates for each instruction an attribute of depth and a attribute of occupation.

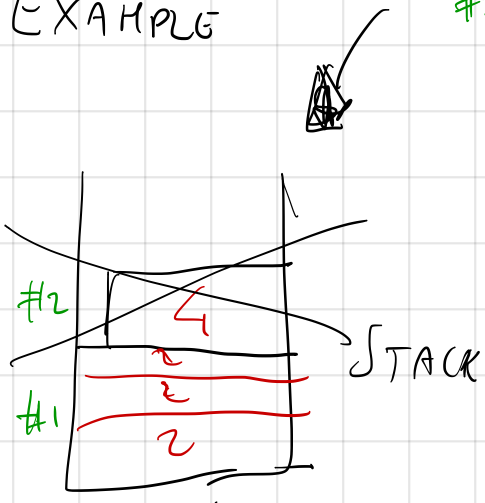
The depth is the number of blocks in which the instruction is enclosed.

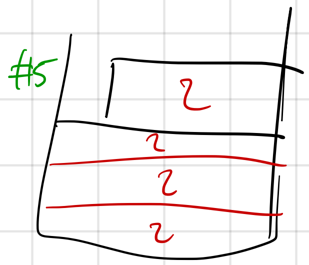
*(space can be reused when block are closed)

The occupation for blocks is the maximum number of bytes needed to host the variables defined in the block (and sub blocks)*
Considering that integers require 2 bytes and real 4 bytes.

EXAMPLE

```
#1 begin var a, b, c : integer; #2 S;
     #3 begin var d: real; S' #4
        end;
     #5 begin var e: integer; S'' #6 ; S''' #6
        end;
   end;
#8 S''''
```





S, S', S'', S''', S'''' are sequences of instructions that are not blocks.

#1: depth = 0 , occ = 10 #2: 1,0 #3: 1,4 #4 2,0 #5: 1,2
#6: 2,0 #7: 2,0 #8: 0,0

Prog → Block

Block → $\underline{\text{begin}}$ Decl ; Com RestOfProg $\underline{\text{end}}$

RestOfProg → ; Com RestOfProg | ε   for avoiding left-recursion

Com → Block

Com → Stat   not block

Decl → $\underline{\text{var}}$ Id ListId : Type   for avoiding left recursion

ListId → , Id ListId | ε

Type → $\underline{\text{integer}}$ | $\underline{\text{real}}$