4 Example exam questions

- Omvandla uttrycket **a + b * (c – d) – e / (f + g / h)** från infix till postfix med hjälp av en stack och **<u>visa varje steg</u>** i processen. **(5p)**
- Visa sedan med hjälp av en stack hur man skulle beräkna det postfix uttryckets värde om **a = 4, b = 3, c = 5, d = 8, e=3, f = 5, g = 7 och h = 9.** **(5p)**
- Visa hur man skulle kunna omvandla uttrycket i del (1) från infix till postfix med hjälp av ett träd. **(5p)**
- Använd samma exempel för att förklara hur man skulle kunna få tillbaka uttrycket i infix notation från postfix. **(5p)**

First Principles & Rules of Thumb

- Infix => postfix
  - operands appear in the same order (in => post)
  - in the corresponding expression tree, operands are always **LEAF NODES**
  - if you forget the precedence rules, work from first principles (see next slide)
  - **ALWAYS** double check your work
    - e.g. Infix => postfix => tree => infix (in-order)

First Principles - examples

- a + b * c
  - x + y => x y +
  - x * y => x y *
  - hence by substitution

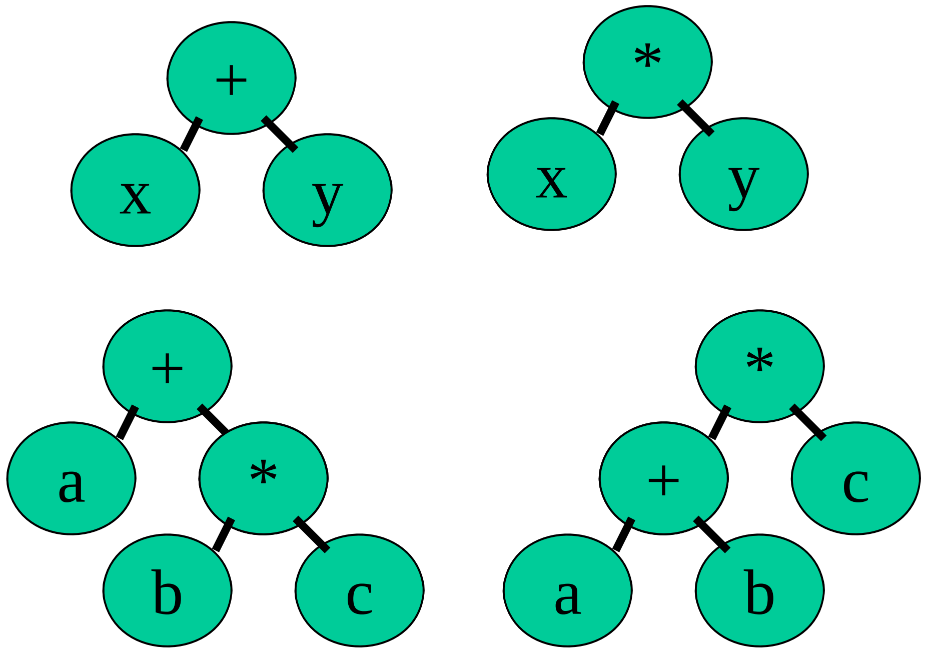
    x = a, y = b * c

    => a b c * +
- (a + b) * c

    x = a + b, y = c

    => a b + c *

- In pictures



Algorithm: infix => postfix

- Input = operand => output operand (=> operand order always same)
- Input = operator => check precedence
  - precedence (input operator) > precedence (tos operator)
    - stack input operator
  - precedence (input operator) <= precedence (tos operator)
    - (pop and output tos operator)$^+$ then stack input operator
- Input = (
  - stack (
- Input =)
  - pop & output tos operator until (pop (
- Input = # (empty) => pop & output tos operator until stack = ¤

Question 1

- Omvandla uttrycket a + b * (c – d) – e / (f + g / h) från infix till postfix med hjälp av en stack och visa varje steg i processen. (5p)
- approach
  - decide format e.g. input string / output string / stack (tos on rhs) / rule
  - go through the example stepwise
- start set-up

input :	a + b * (c – d) – e / (f + g / h) #
output:	¤
stack:	¤
rule:	§ <text>

Answer 1 (input ; output ; stack ; rule)

a + b * (c – d) – e / (f + g / h) #	; ¤	; ¤	; start set-up
+ b * (c – d) – e / (f + g / h) #	; a	; ¤	; output a
b * (c – d) – e / (f + g / h) #	; a	; +	; stack +
* (c – d) – e / (f + g / h) #	; a b	; +	; output b
(c – d) – e / (f + g / h) #	; a b	; + *	; stack * § **prec * > +**
c – d) – e / (f + g / h) #	; a b	; + * (	; stack (§ always
– d) – e / (f + g / h) #	; a b c	; + * (	; output c
d) – e / (f + g / h) #	; a b c	; + * (–	; stack - § **prec - > (**
) – e / (f + g / h) #	; a b c d	; + * (–	; output d
– e / (f + g / h) #	; a b c d -	; + *	; output - § pop to (
– e / (f + g / h) #	; a b c d - *	; +	; output * § **prec * > -**

Answer 1 (input ; output ; stack ; rule) (continued)

– e / (f + g / h) #	; a b c d - *	; +	; output * § **prec * > -**
– e / (f + g / h) #	; a b c d - * +	; ¤	; output + § **prec + = -**
e / (f + g / h) #	; a b c d - * +	; –	; stack - § ¤ stack
/ (f + g / h) #	; a b c d - * + e	; –	; output e
(f + g / h) #	; a b c d - * + e	; – /	; stack / § **prec / > -**
f + g / h) #	; a b c d - * + e	; – / (	; stack (§ always
+ g / h) #	; a b c d - * + e f	; – / (	; output f
g / h) #	; a b c d - * + e f	; – / (+	; stack - § **prec + > (**
/ h) #	; a b c d - * + e f g	; – / (+	; output g
h) #	; a b c d - * + e f g	; – / (+ /	; stack / § **prec / > +**
) #	; a b c d - * + e f g h	; – / (+ /	; output h

Answer 1 (input ; output ; stack ; rule) (continued)

) #	; a b c d - * + e f g h	; – / (+ /	; output h
#	; a b c d - * + e f g h / +	; – /	; output § pop to (
#	; a b c d - * + e f g h / + / –	; ¤	; output - eos # pop

result: a b c d - * + e f g h / + / – ; eos # / stack ¤

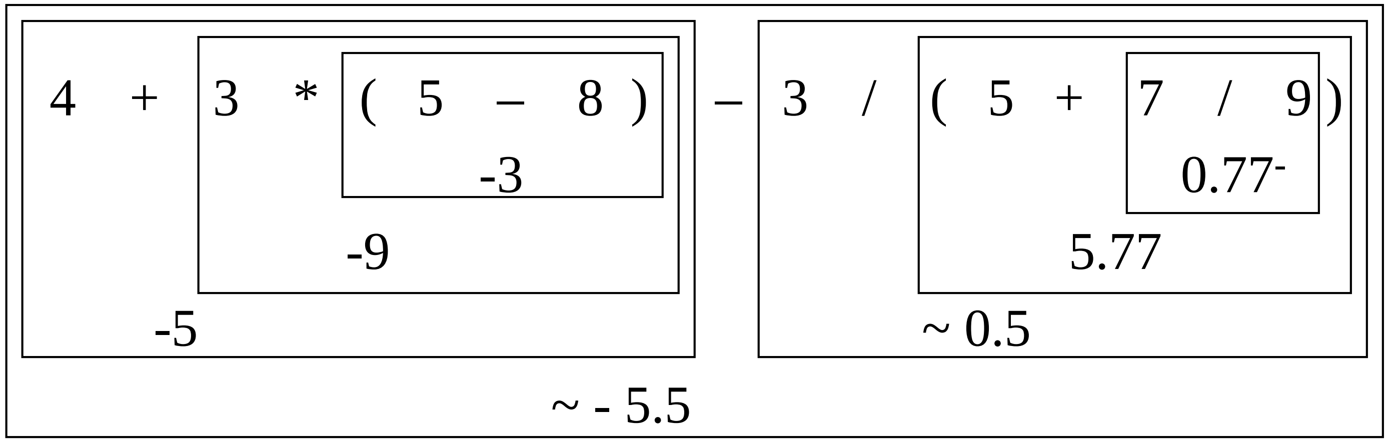
input: a + b * (c – d) – e / (f + g / h)

cross check: $\boxed{\boxed{a \quad + \quad \boxed{b \quad * \quad \boxed{(c - d)}}} \quad - \quad \boxed{e \quad / \quad (\boxed{f \quad + \quad \boxed{g / h}})}}$

Question 2

- Visa sedan med hjälp av en stack hur man skulle beräkna det postfix uttryckets värde om **a = 4, b = 3, c = 5, d = 8, e=3, f = 5, g = 7 och h = 9.** **(5p)**

- infix: a + b * (c – d) – e / (f + g / h)



Answer 2

- Approach
  - postfix: a b c d - * + e f g h / + / –
  - substitute: 4 3 5 8 - * + 3 5 7 9 / + / -
- start set-up
  - input: 4 3 5 8 - * + 3 5 7 9 / + / -
  - stack: ¤ (tos on rhs)
  - rule: (see below)
- algorithm
  - if input = operand, stack operand
  - if input = operator, apply operator to tos & tos_{-1}: **tos_{-1} op tos**

Answer 2 (input ; stack ; rule) (continued)

4 3 5 8 - * + 3 5 7 9 / + / -	;	¤	;	start set-up
3 5 8 - * + 3 5 7 9 / + / -	;	4	;	stack 4
5 8 - * + 3 5 7 9 / + / -	;	4 3	;	stack 3
8 - * + 3 5 7 9 / + / -	;	4 3 5	;	stack 5
- * + 3 5 7 9 / + / -	;	4 3 5 8	;	stack 8
* + 3 5 7 9 / + / -	;	4 3 -3	;	apply - **tos$_{-1}$ op tos**
+ 3 5 7 9 / + / -	;	4 -9	;	apply * **tos$_{-1}$ op tos**
3 5 7 9 / + / -	;	-5	;	apply + **tos$_{-1}$ op tos**

Answer 2 (input ; stack ; rule) (continued)

3 5 7 9 / + / -	; -5	; apply + **tos_{-1} op tos**
5 7 9 / + / -	; -5 3	; stack 3
7 9 / + / -	; -5 3 5	; stack 5
9 / + / -	; -5 3 5 7	; stack 7
/ + / -	; -5 3 5 7 9	; stack 9
+ / -	; -5 3 5 0.77	; apply / **tos_{-1} op tos**
/ -	; -5 3 5.77	; apply + **tos_{-1} op tos**
-	; -5 ~0.5	; apply / **tos_{-1} op tos**
¤	; ~ -5.5	; apply - **tos_{-1} op tos**

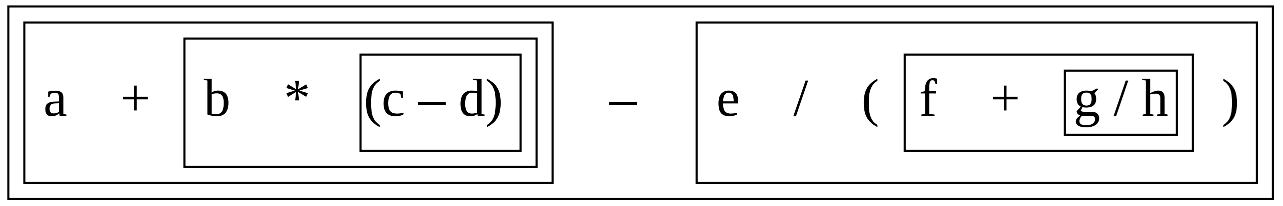
Question 3

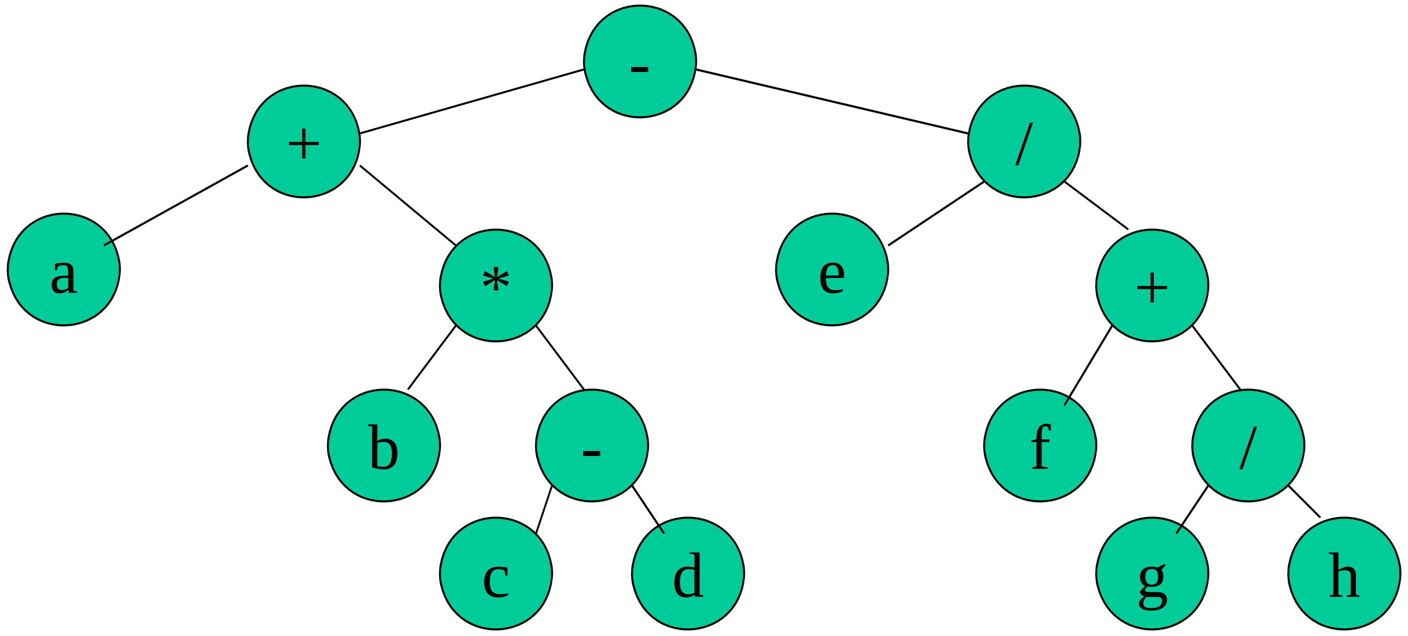
- Visa hur man skulle kunna omvandla uttrycket i del (1) från infix till postfix med hjälp av ett träd. **(5p)**

- approach
  - infix: a + b * (c – d) – e / (f + g / h)
  - postfix: a b c d - * + e f g h / + / –
- from previous answer (answer 1 to question 1)

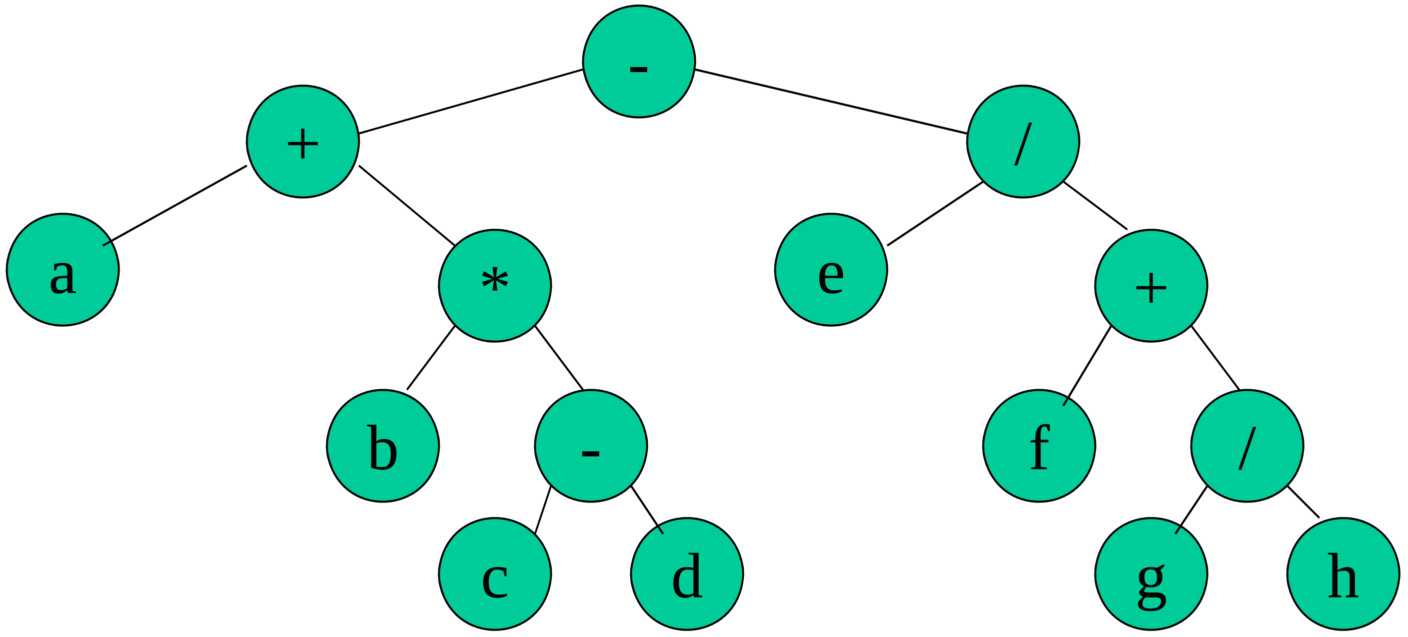
$$\boxed{\boxed{a + \boxed{b * \boxed{(c - d)}}} - \boxed{e \;/\; (\boxed{f + \boxed{g / h}})}}$$

From first principles





Post-order traversal of tree



Answer to question 3:   a b c d - * + e f g h / + / –

Question 4

- Använd samma exempel för att förklara hur man skulle kunna få tillbaka uttrycket i infix notation från postfix.
  (5p)

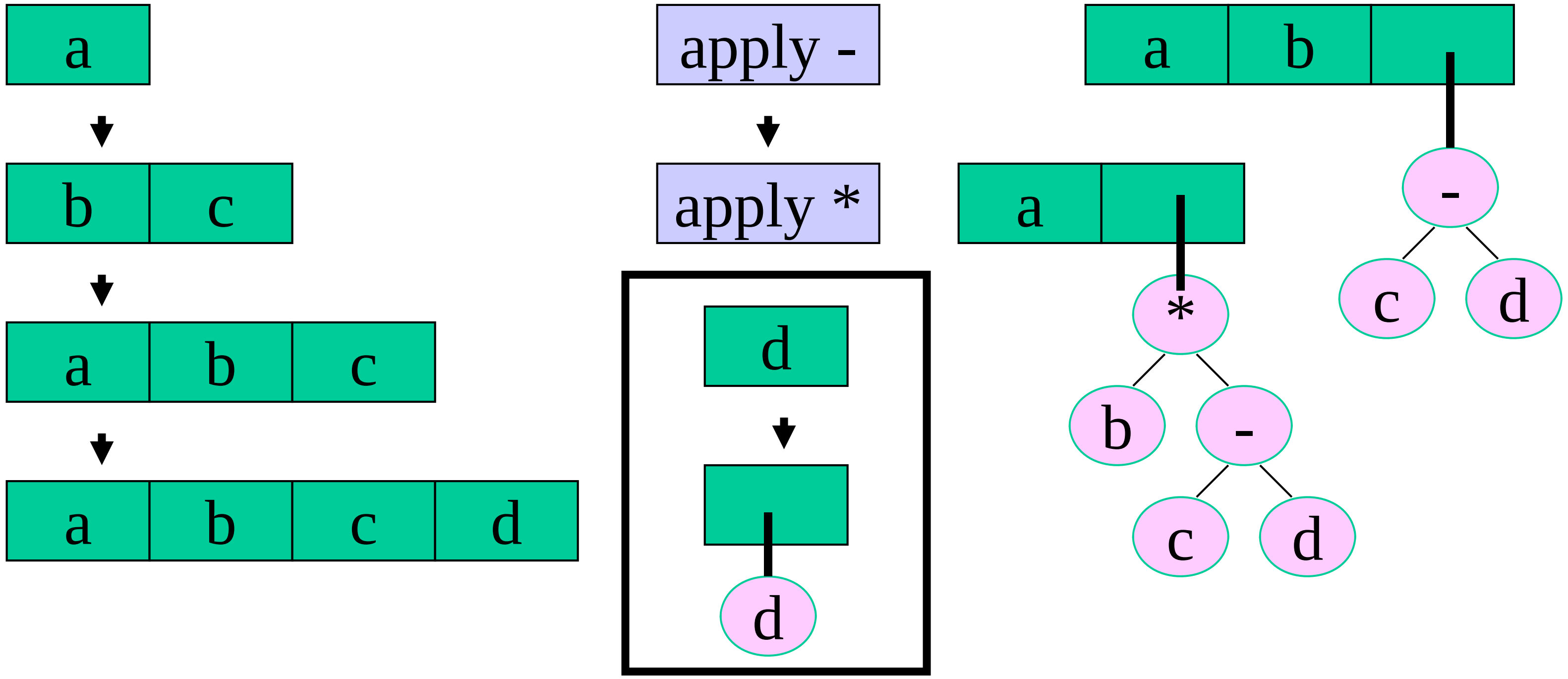
- Approach
  - reconstruct the tree from the postfix expression
  - perform an in-order traversal of the tree to obtain the infix form of the expression
  - Comment: think about how you “restore” the “(“ and “)”
  - HINT: think about the precedence of the operators

Postfix => tree

Postfix:	a b c d - * + e f g h / + / –
Method:	apply same rules as in question 2
Algorithm:	if input = operand, stack operand if input = operator, apply operator to tos & tos$_{-1}$: tos$_{-1}$ op tos to make a tree
Comment:	this is a generalisation of the calculation in answer 2 - instead of a particular value, the result is a (sub-)tree and the final result is a tree for the whole expression

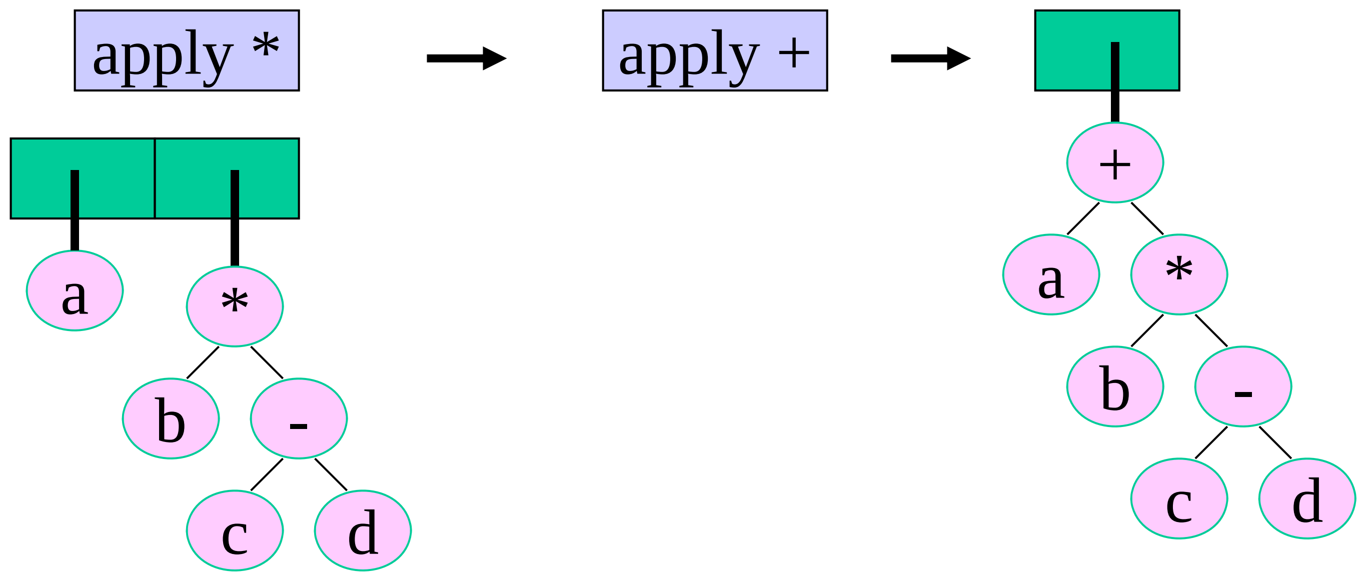
Postfix => tree: example

- Postfix: a b c d - * + e f g h / + / –



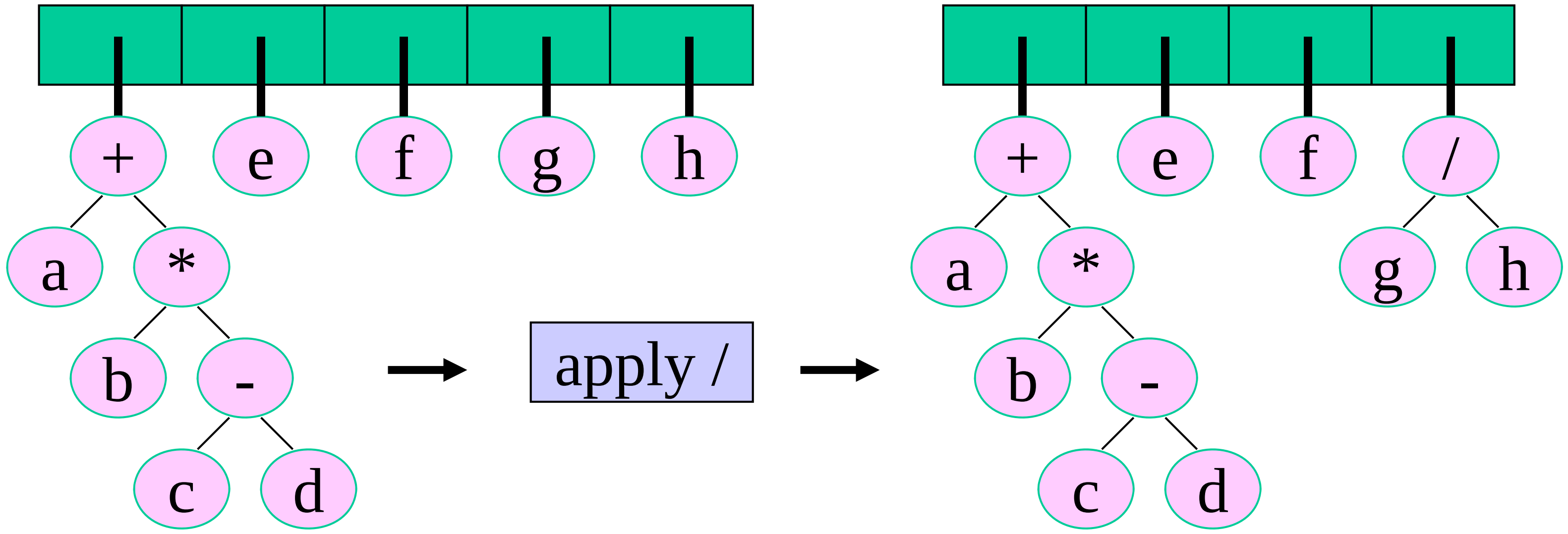
Postfix => tree: example (continued)

- Postfix: a b c d - * + e f g h / + / –



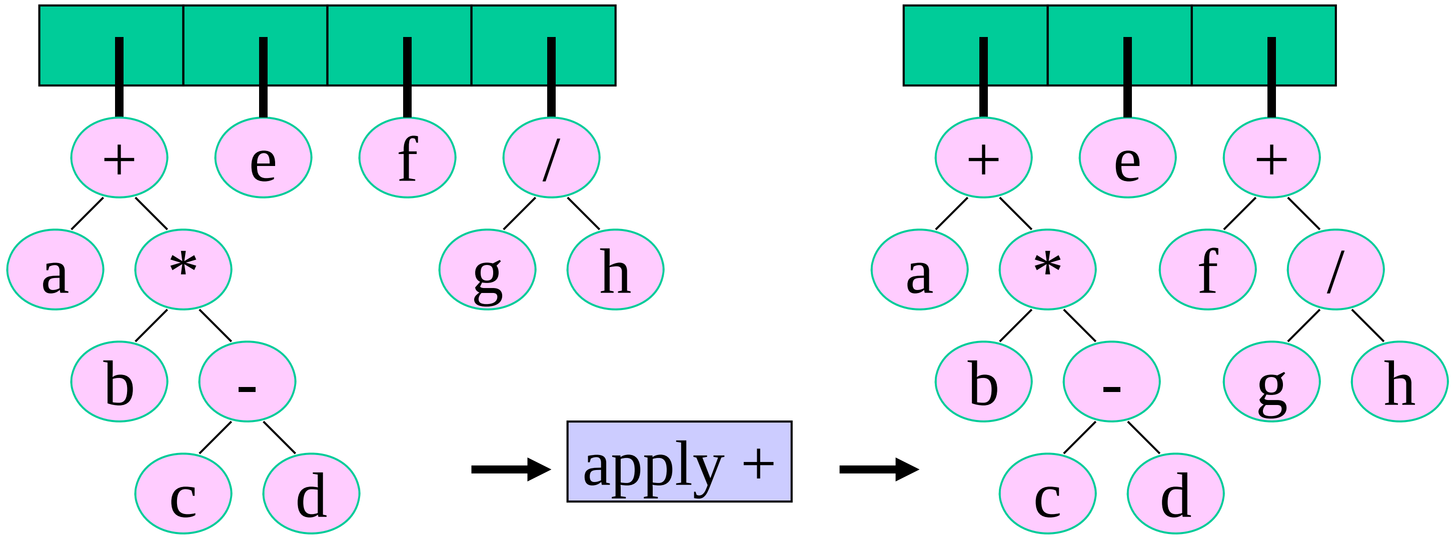
Postfix => tree: example (continued)

- Postfix: a b c d - * + e f g h / + / –



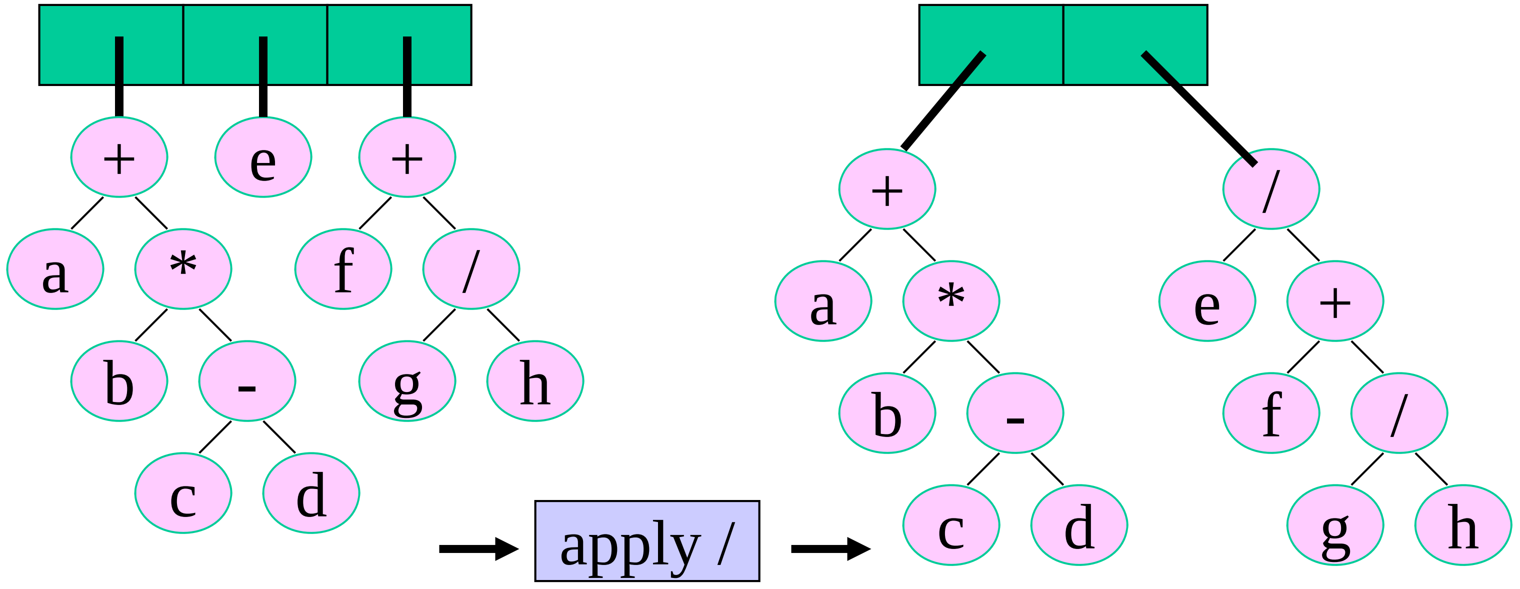
Postfix => tree: example (continued)

- Postfix: a b c d - * + e f g h / + / –



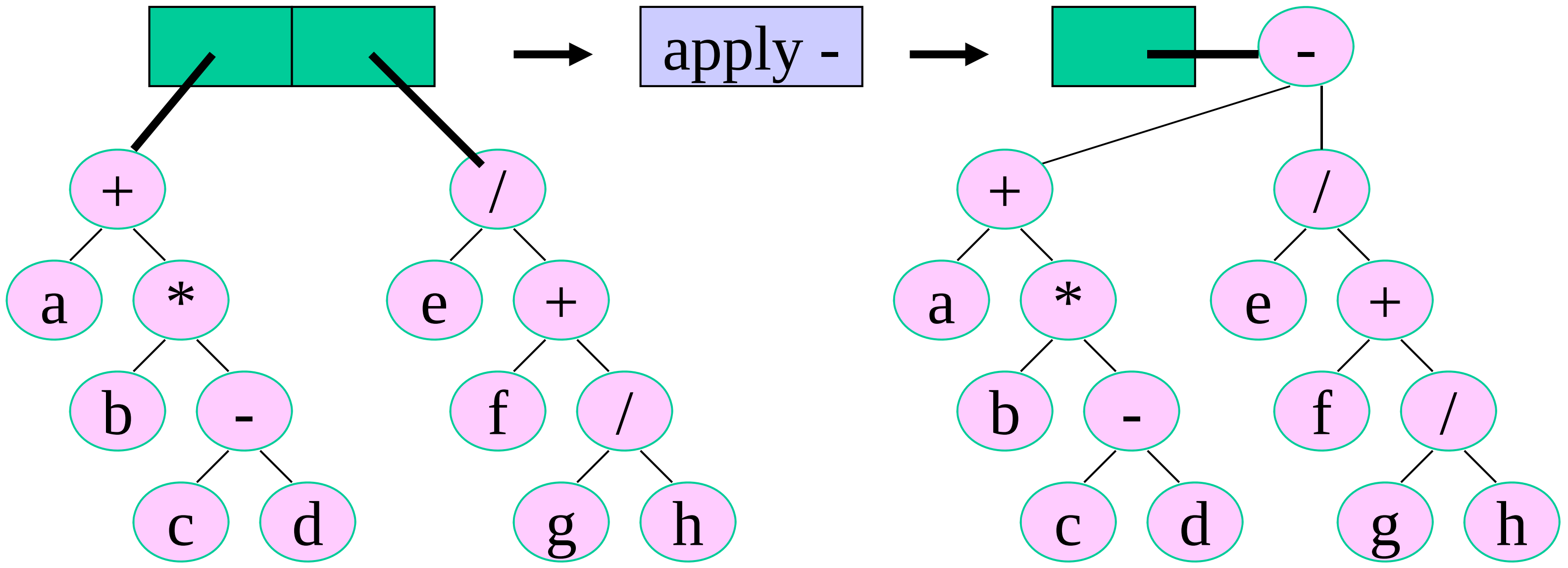
Postfix => tree: example (continued)

- Postfix: a b c d - * + e f g h / + / –



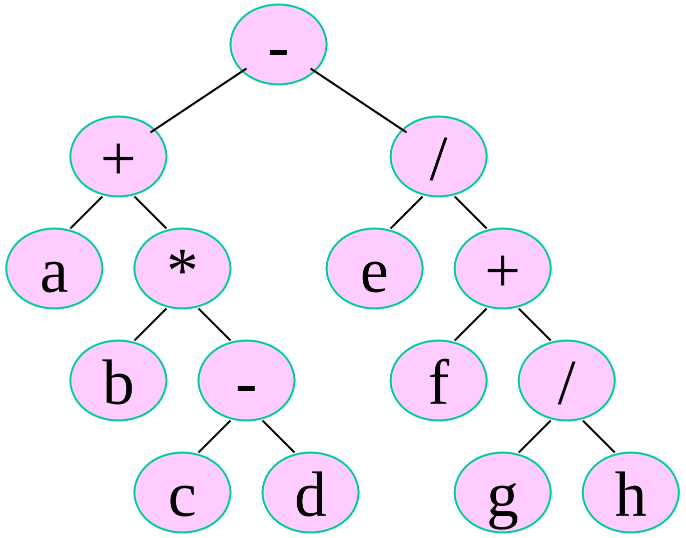
Postfix => tree: example (continued)

- Postfix: a b c d - * + e f g h / + / –



Postfix => tree: example (continued)

- Postfix: a b c d - * + e f g h / + / –



now cross check

in-order traversal:

a + b * (c – d) – e / (f + g / h)

post-order traversal:

a b c d - * + e f g h / + / –

hence answer to question 4