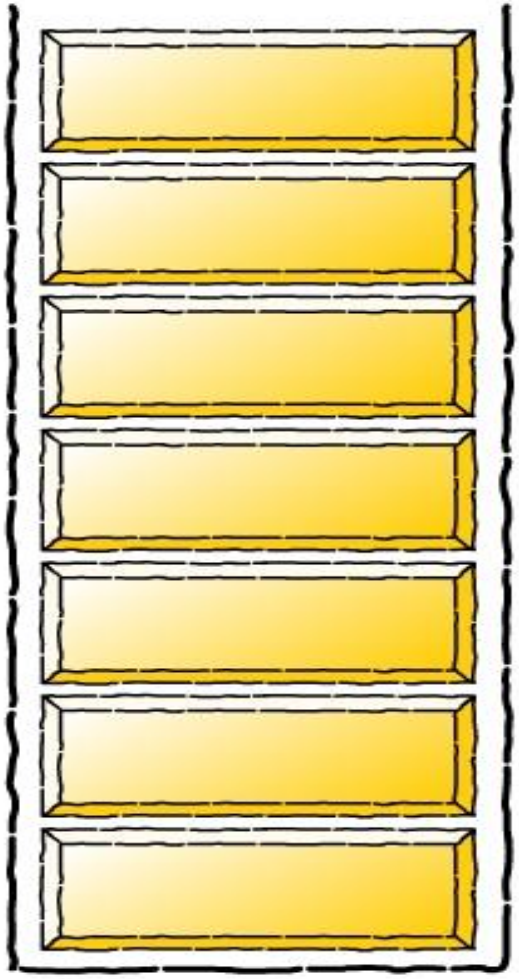
Infix to postfix conversion



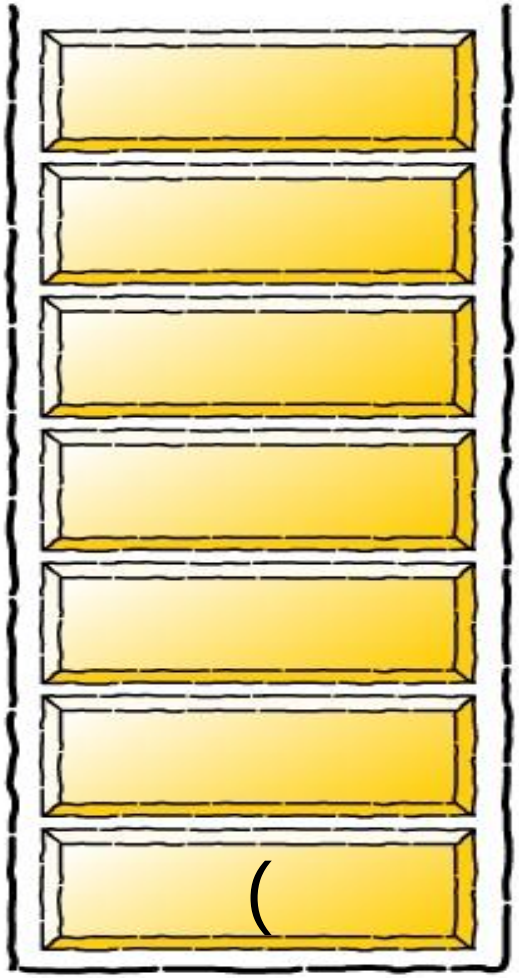
infixVect

(a + b - c) * d – (e + f)

postfixVect

Infix to postfix conversion

stackVect



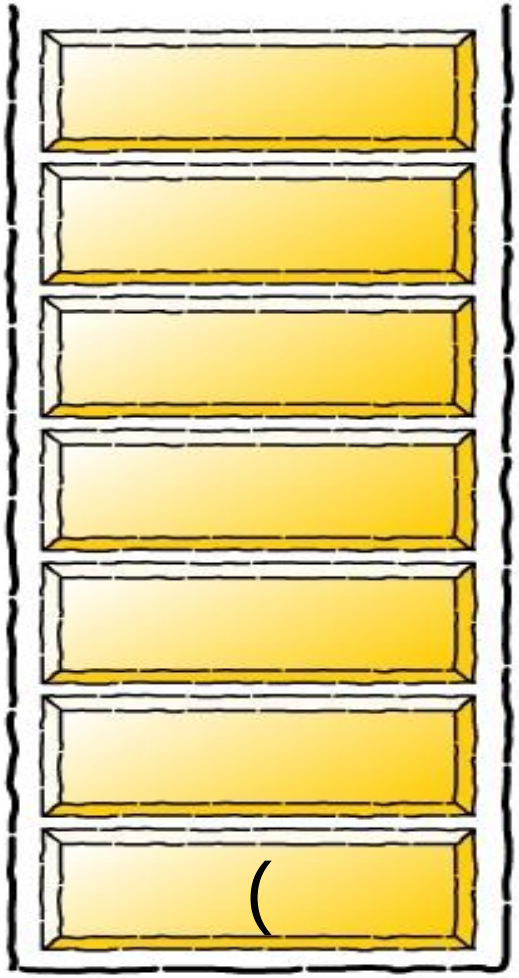
infixVect

a + b - c) * d – (e + f)

postfixVect

Infix to postfix conversion

stackVect



infixVect

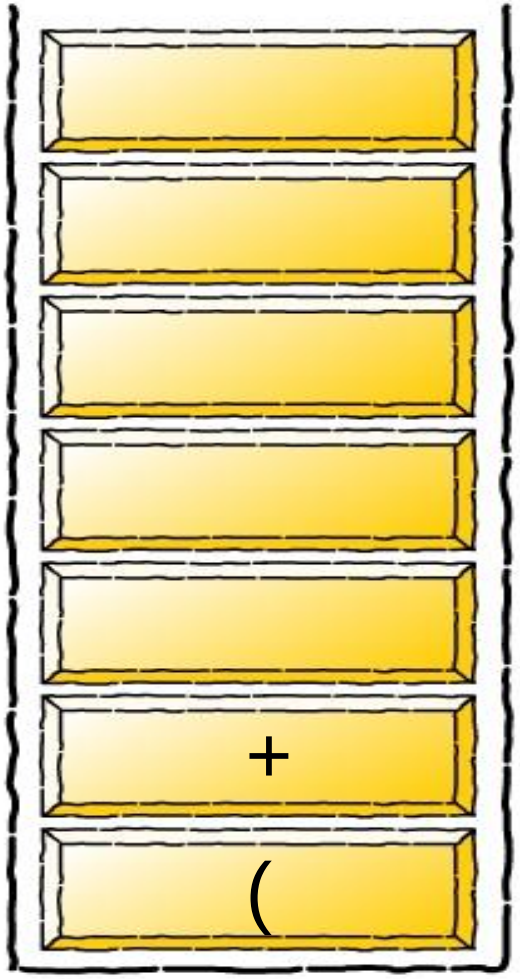
+ b - c) * d – (e + f)

postfixVect

a

Infix to postfix conversion

stackVect



infixVect

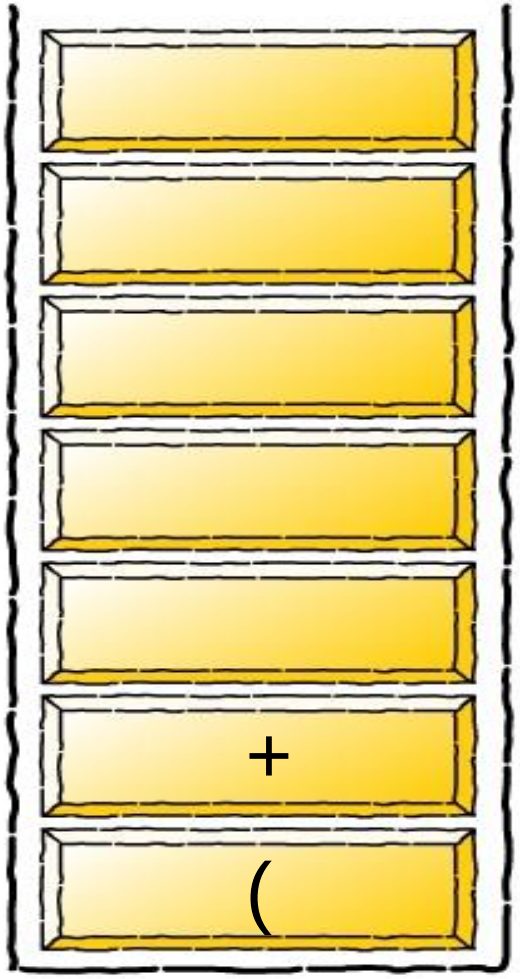
b - c) * d – (e + f)

postfixVect

a

Infix to postfix conversion

stackVect



infixVect

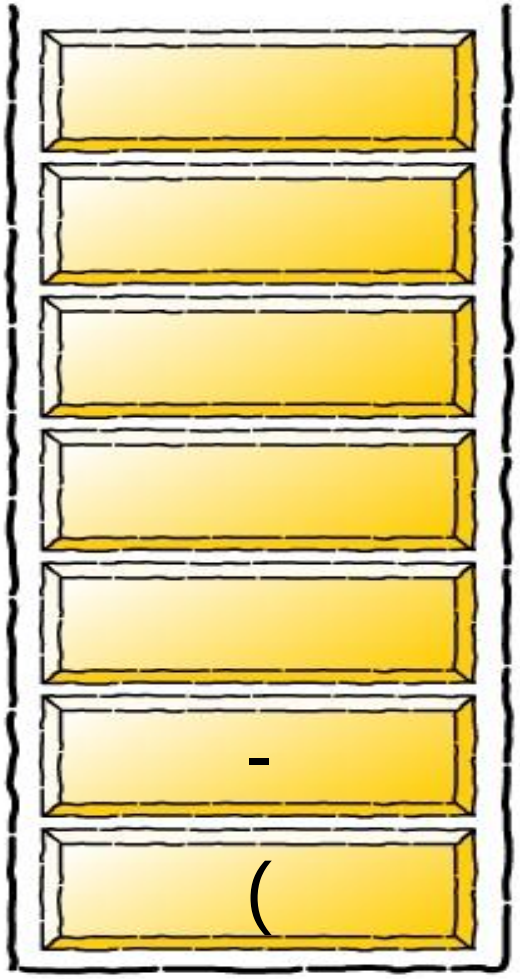
- c) * d – (e + f)

postfixVect

a b

Infix to postfix conversion

stackVect



infixVect

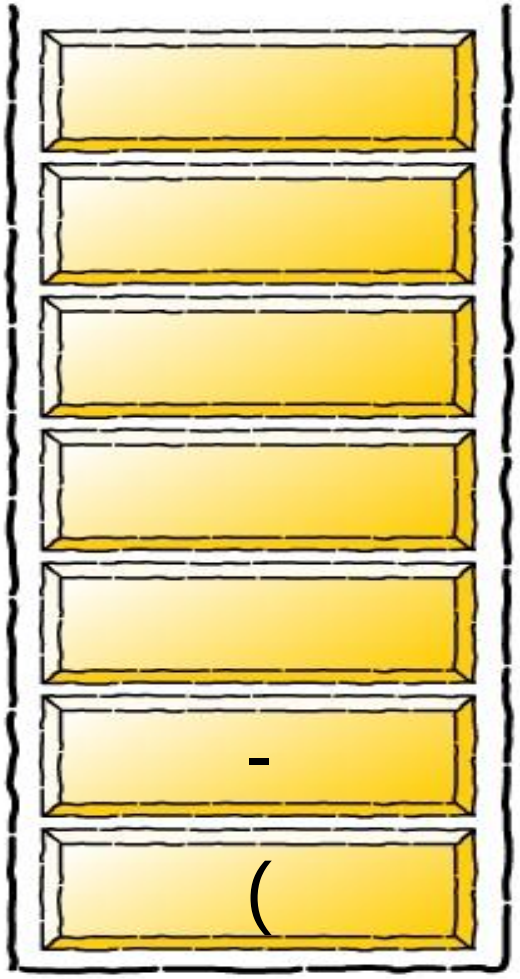
c) * d – (e + f)

postfixVect

a b +

Infix to postfix conversion

stackVect



infixVect

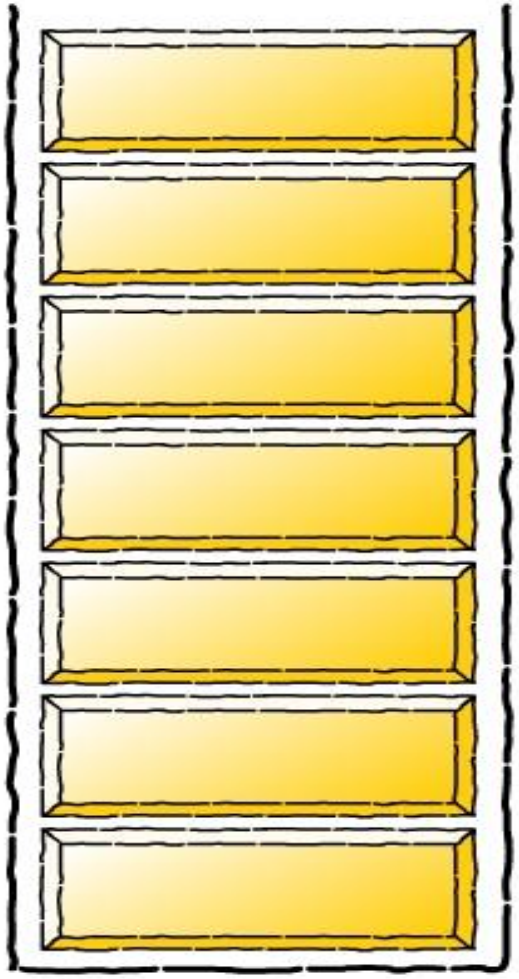
) * d – (e + f)

postfixVect

a b + c

Infix to postfix conversion

stackVect



infixVect

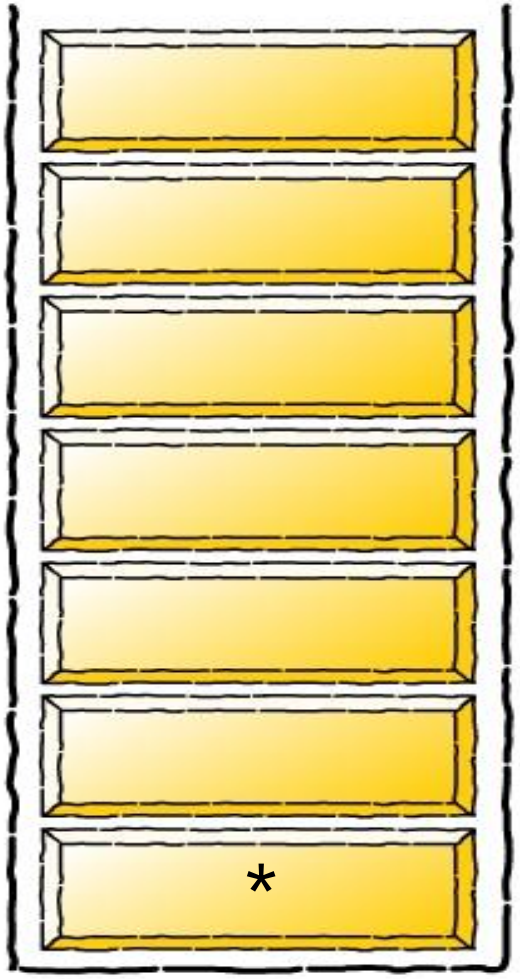
* d – (e + f)

postfixVect

a b + c -

Infix to postfix conversion

stackVect



infixVect

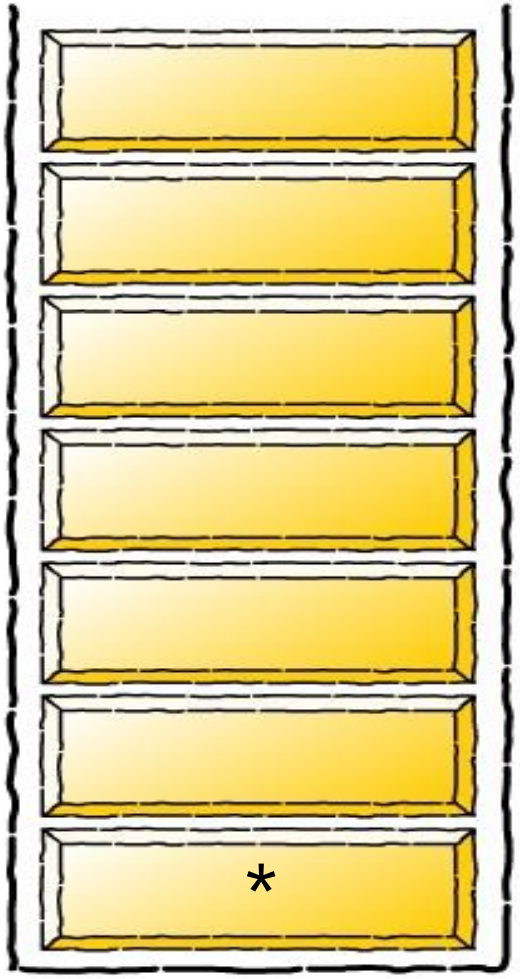
d – (e + f)

postfixVect

a b + c -

Infix to postfix conversion

stackVect



infixVect

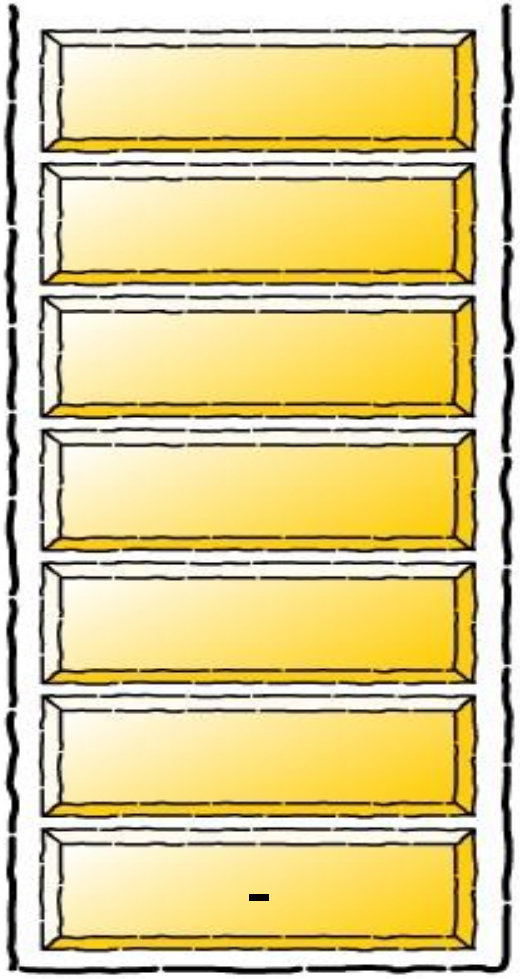
– (e + f)

postfixVect

a b + c - d

Infix to postfix conversion

stackVect



infixVect

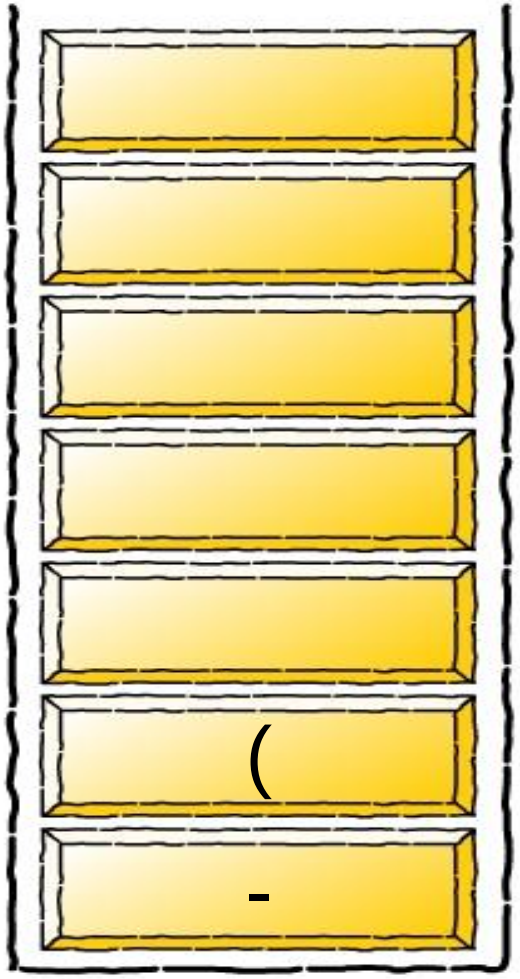
(e + f)

postfixVect

a b + c – d *

Infix to postfix conversion

stackVect



infixVect

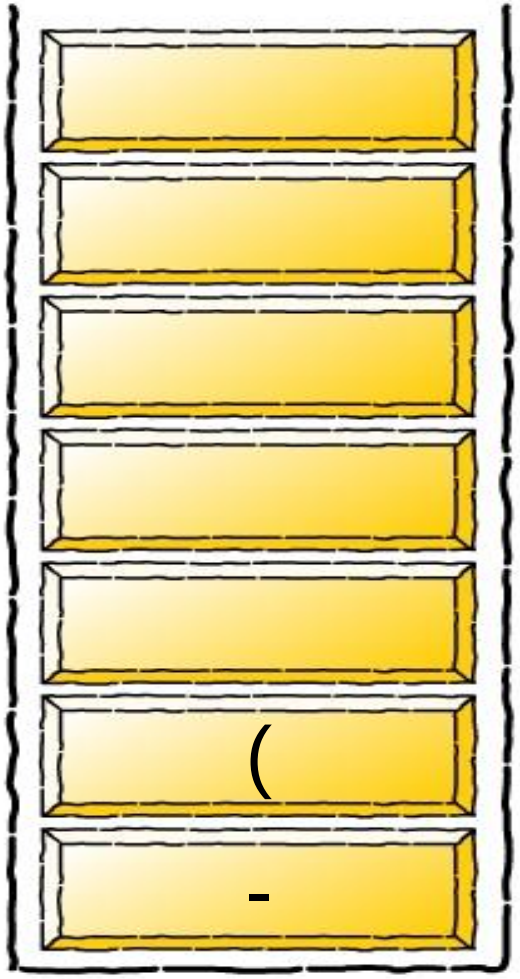
e + f)

postfixVect

a b + c – d *

Infix to postfix conversion

stackVect



infixVect

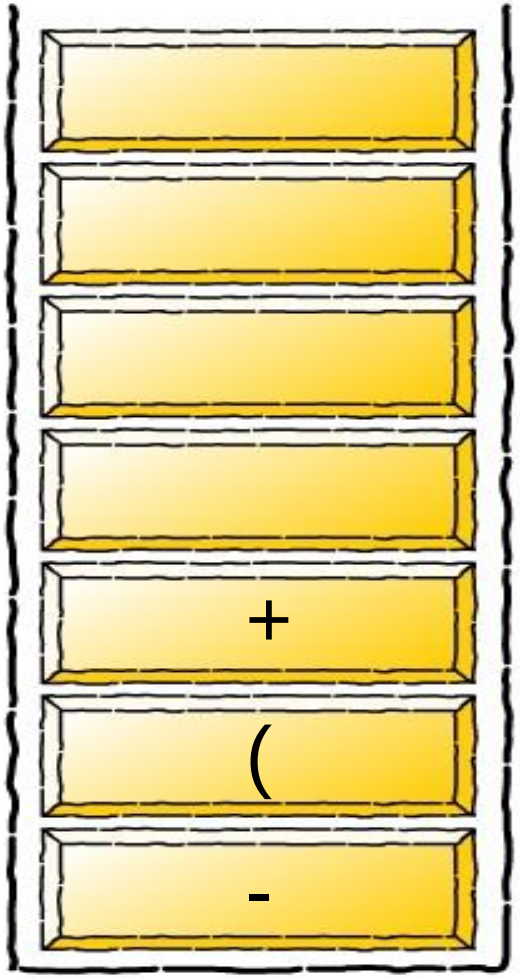
+ f)

postfixVect

a b + c – d * e

Infix to postfix conversion

stackVect



infixVect

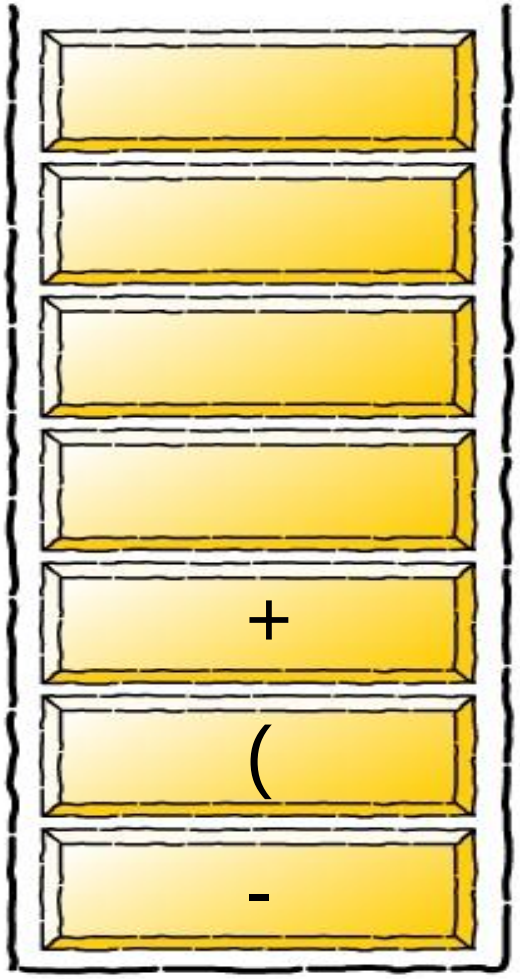
f)

postfixVect

a b + c – d * e

Infix to postfix conversion

stackVect



infixVect

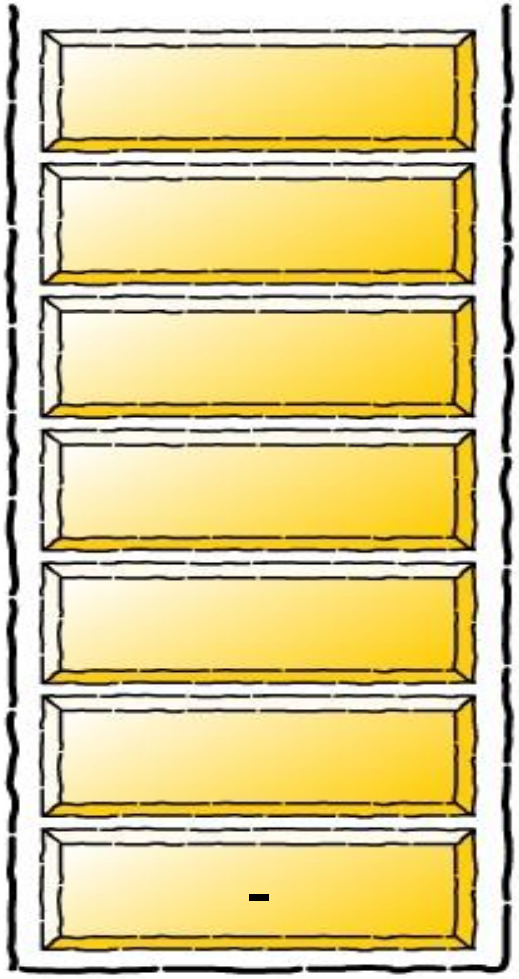
)

postfixVect

a b + c – d * e f

Infix to postfix conversion

stackVect



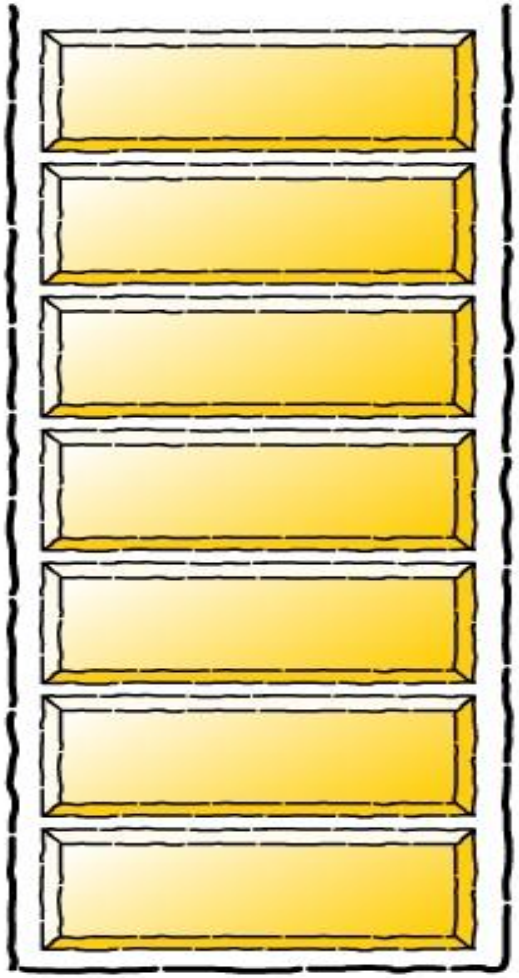
infixVect

postfixVect

a b + c – d * e f +

Infix to postfix conversion

stackVect



infixVect

postfixVect

a b + c – d * e f + -

Infix to postfix conversion

Use a loop to read the tokens one by one from a vector infixVect of tokens (strings) representing an infix expression.

For each token do the following in the loop:

- When the *token* is an operand
  - Add it to the end of the vector postfixVect of token (strings) that is used to store the corresponding postfix expression

- When the *token* is a left parenthesis “(”
  - Push_back the token x to the end of the vector stackVect of token (strings) that simulates a stack.
- When the *token* is a right parenthesis “)”
  - Repeatedly pop_back a token y from stackVect and push_back that token y to postfixVect until “(“ is encountered in stackVect. Then pop_back “(“ from stackVect.
  - If stackVect is already empty before finding a “(“, that expression is not a valid expression.
- When the *token* is an operator, see next slide.

Infix to postfix conversion

- When the *token x* is an operator
  - **Write a loop that checks the following conditions:**
    1. The stack stackVect is **not** empty
    2. The token y **currently** in the end of stackVect is an operator. In other words, it is not **not** a lef parenthesis "(" .
    3. y is an operator of **higher or equal** precedence than that of x,
  - **As long as all the three conditions above are true, in the loop above repeatedly do the following in the body of the loop :**
    - Call push_back to store a copy of the token y into postfixVect
    - Call pop_back to remove the token y from stackVect
  - **Note: The loop above will stops as soon as any of the three conditions is not true.**
  - **After the loop,** push_back the token x into stackVect.

Infix to postfix conversion

After the loop (in the previous slide) has processes all the tokens in infixVect and stop,

use another loop to repeatedly do the following as long as the stack vector stackVect is not empty yet:

- Call push_back to store a copy of the token on the top of the stack vector stackVect into postfixVect.
- Call pop_back to remove the top token y from the stack vector.