

Data Structures and Algorithm

Xiaoqing Zheng
zhengxq@fudan.edu.cn



What is data structure?

A data structure is a way to store and organize data in order to facilitate access and modifications.

Elementary data structures

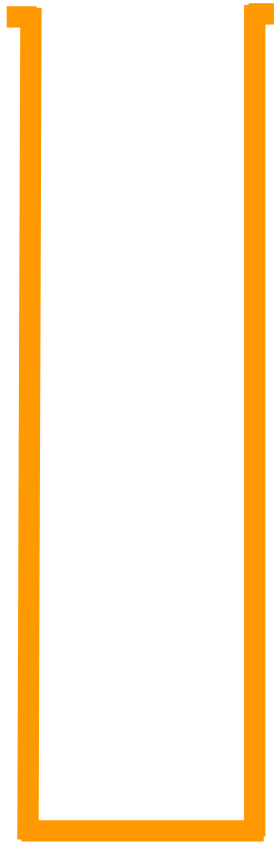
- *Stacks*
- *Queues*
- *Linked lists*
- *Trees*

Stacks

$top[S] = 0$

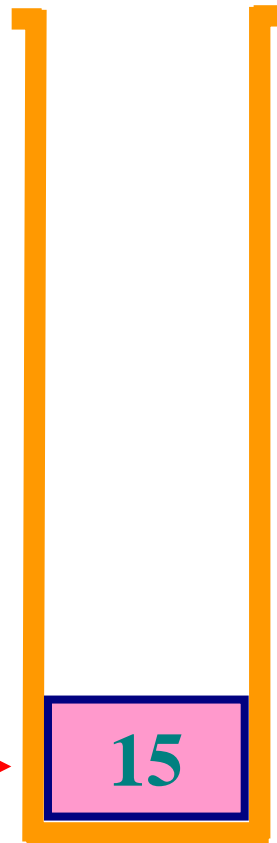


s



Stacks

$top[S] = 1$

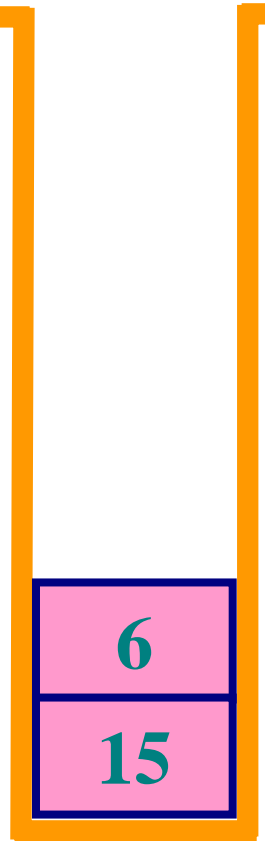


S

PUSH 15

Stacks

$top[S] = 2$

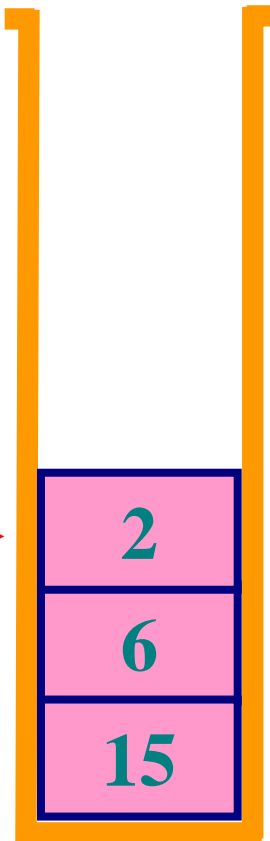


S

PUSH 6

Stacks

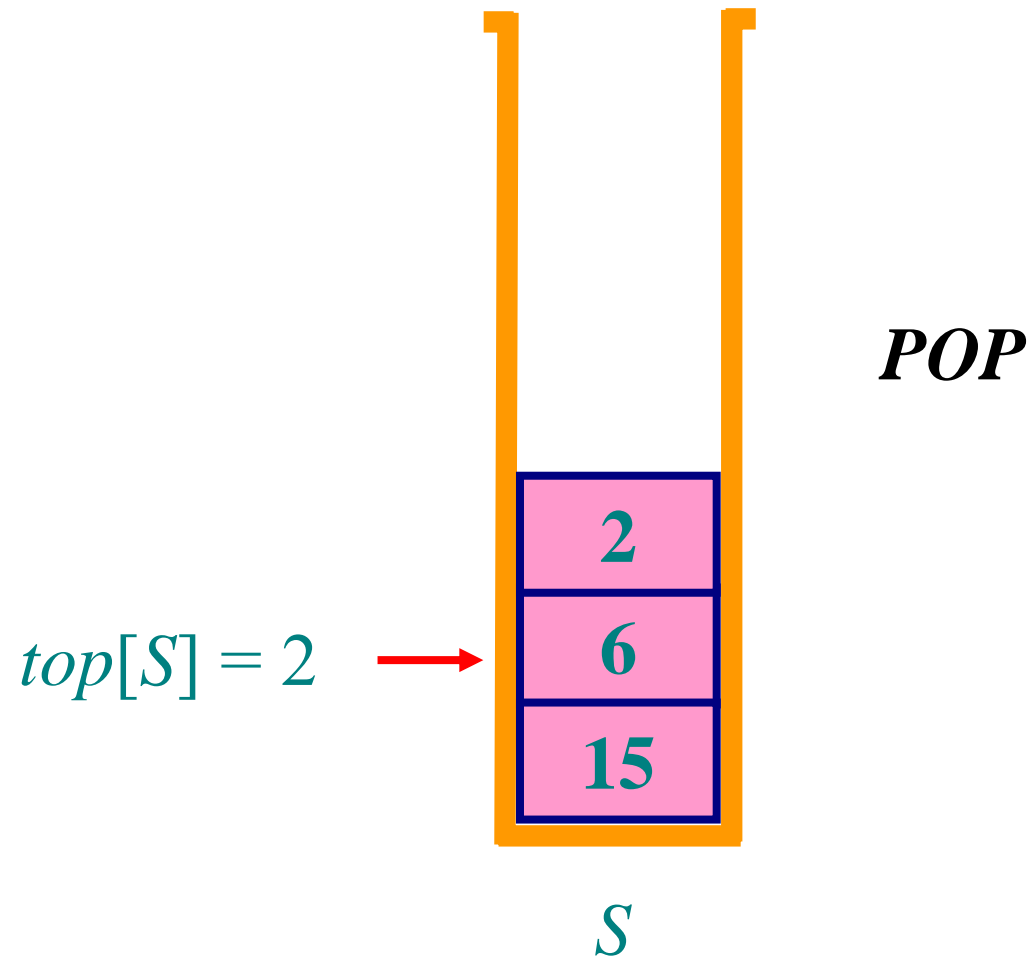
$top[S] = 3$



S

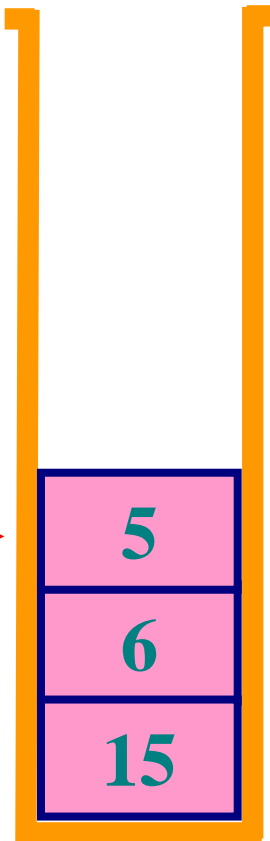
PUSH 2

Stacks



Stacks

$top[S] = 3$

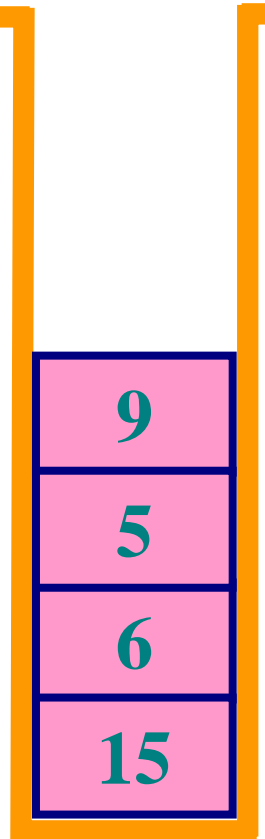


S

PUSH 5

Stacks

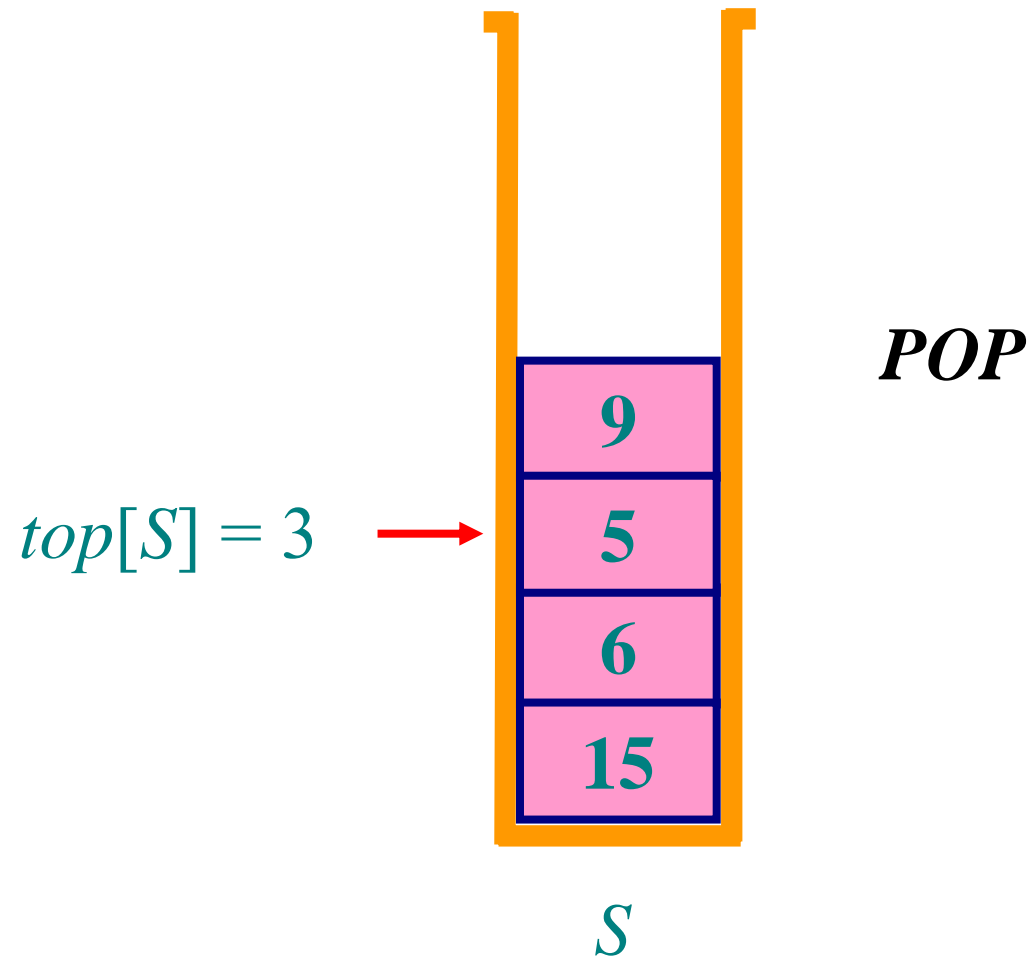
$top[S] = 4$



S

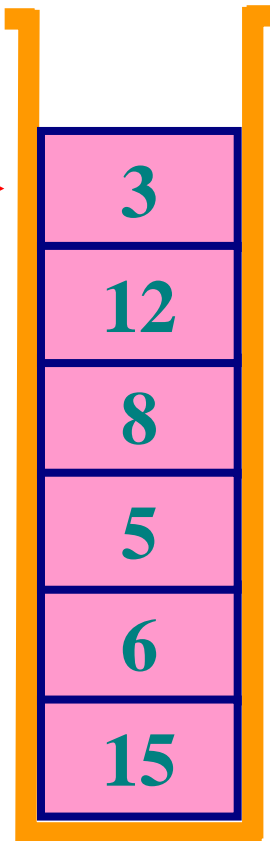
PUSH 9

Stacks



Stacks

$top[S] = 6$



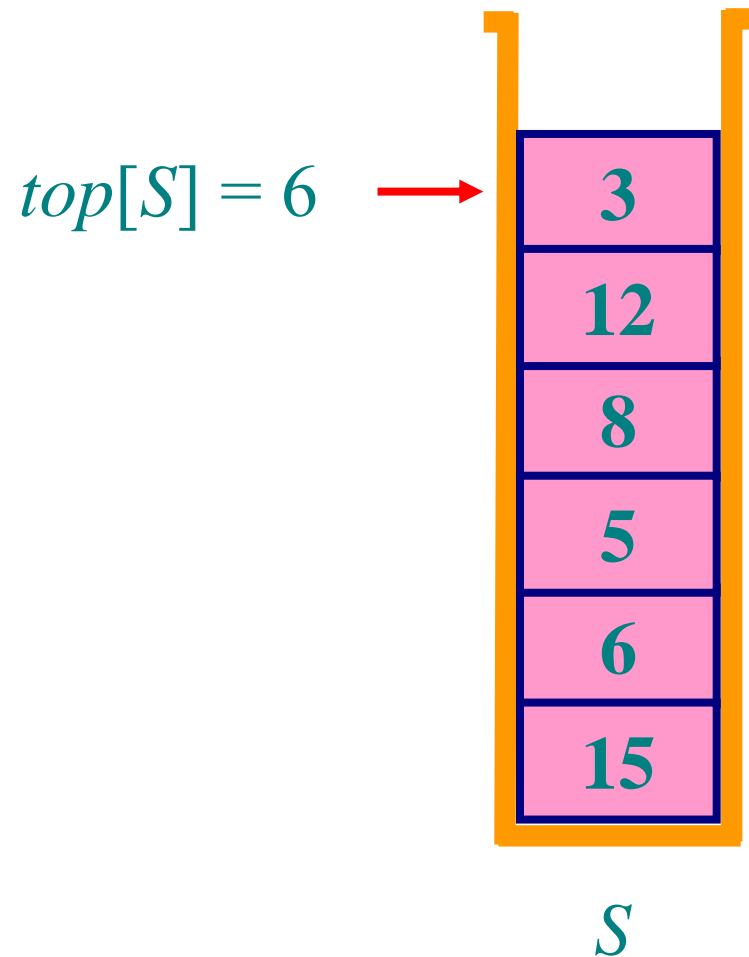
S

PUSH 8

PUSH 12

PUSH 3

Stacks



LIFO

last-in, first-out

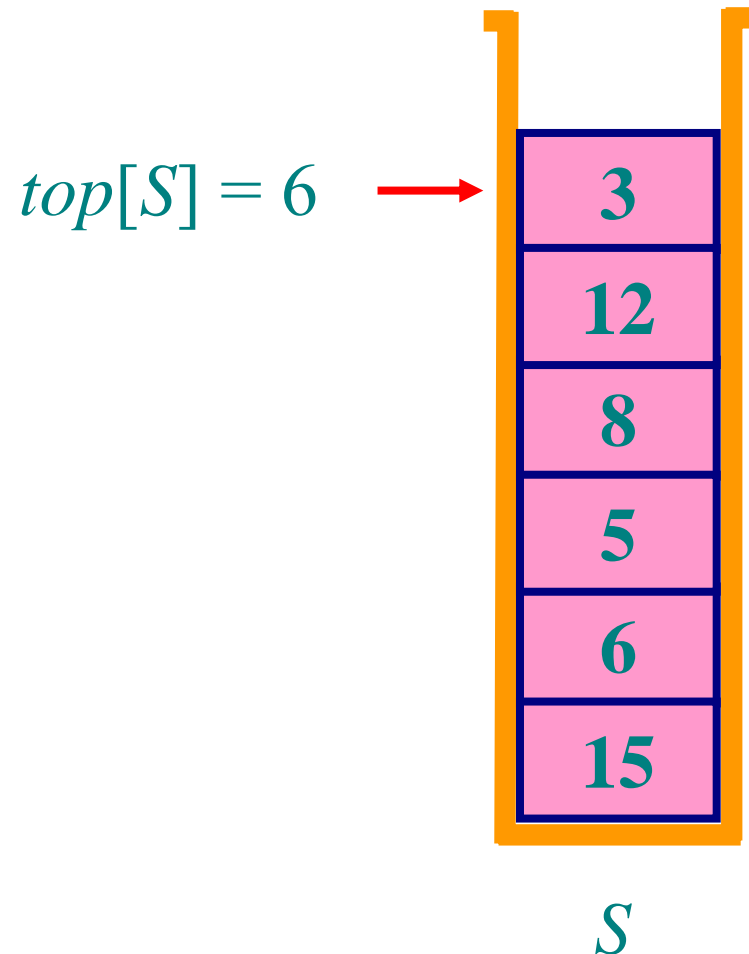
Underflows

If an empty stack is popped.

Overflows

If $top[S]$ exceeds n .

Stacks



STACK-EMPTY(S)

1. **if** $top[S] = 0$
2. **then return** TRUE
3. **else return** FALSE

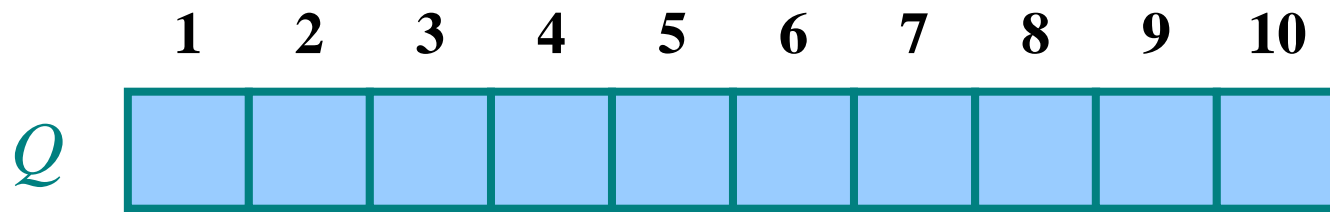
PUSH(S, x)

1. $top[S] \leftarrow top[S] + 1$
2. $S[top[S]] \leftarrow x$

POP(S)

1. **if** STACK-EMPTY(S)
2. **then error** "underflow"
2. **else** $top[S] \leftarrow top[S] - 1$
4. **return** $S[top[S] + 1]$

Queues



$head[Q] = tail[Q] = 1$

Age Group	Percentage
18-24	15%
25-34	25%
35-44	30%
45-54	20%
55-64	10%
65-74	5%
75-84	2%
85+	1%

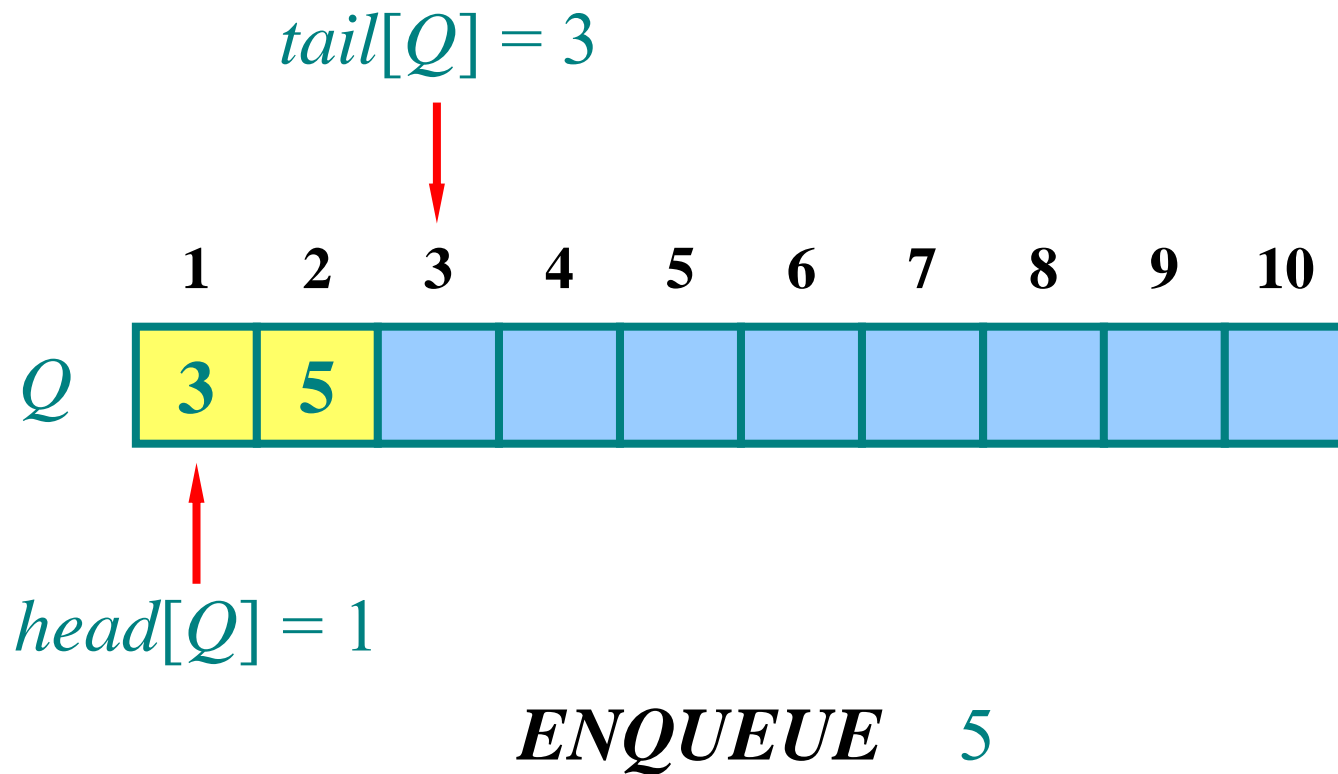


Q

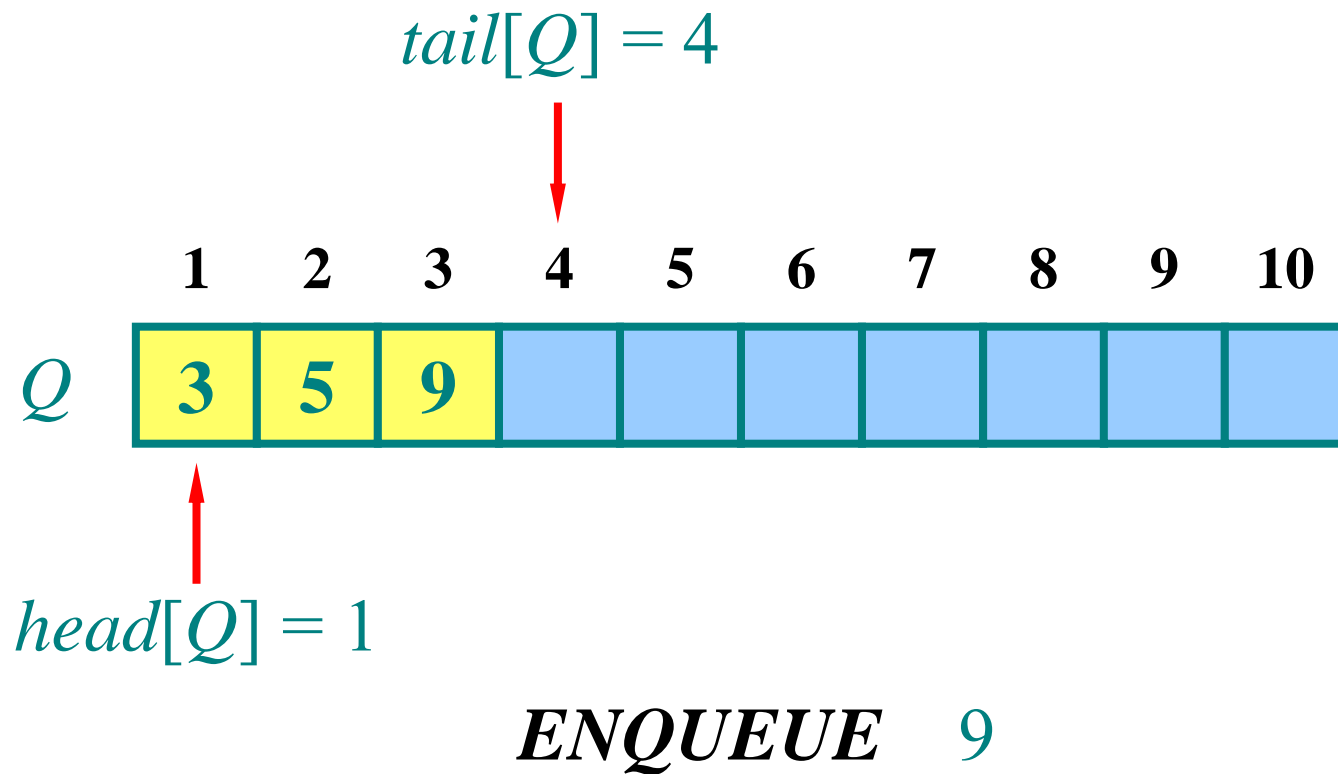


ENQUEUE 3

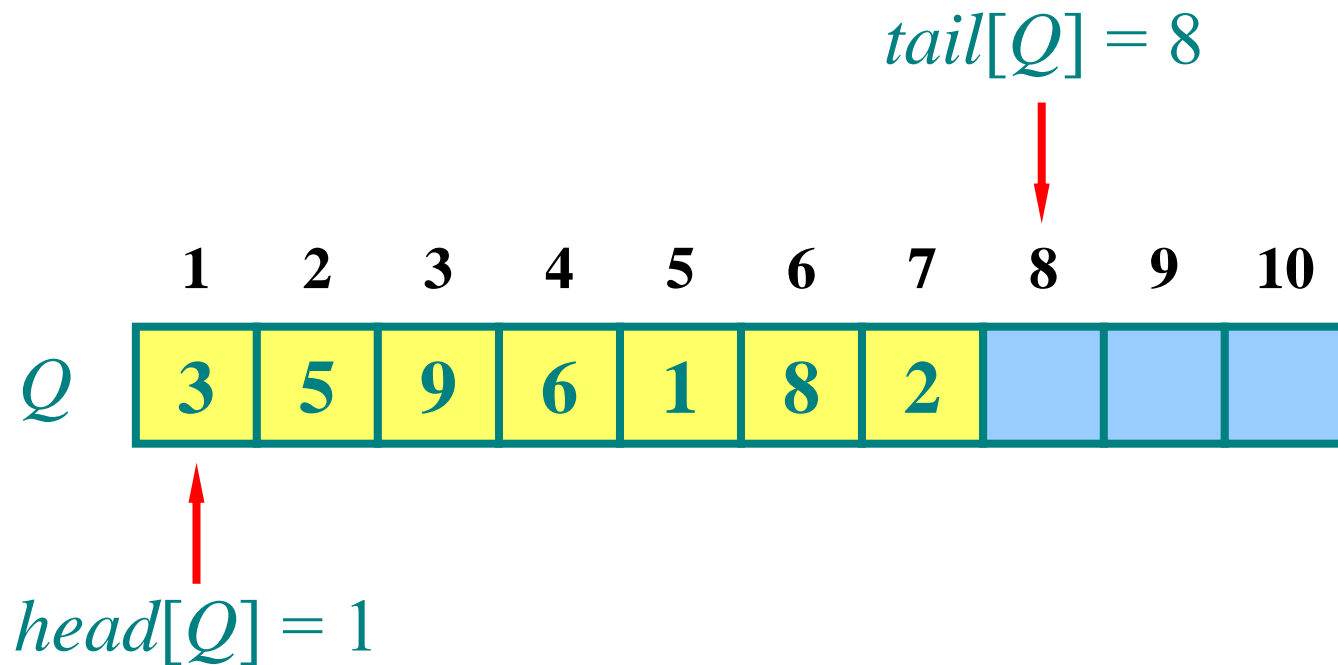
Queues



Queues

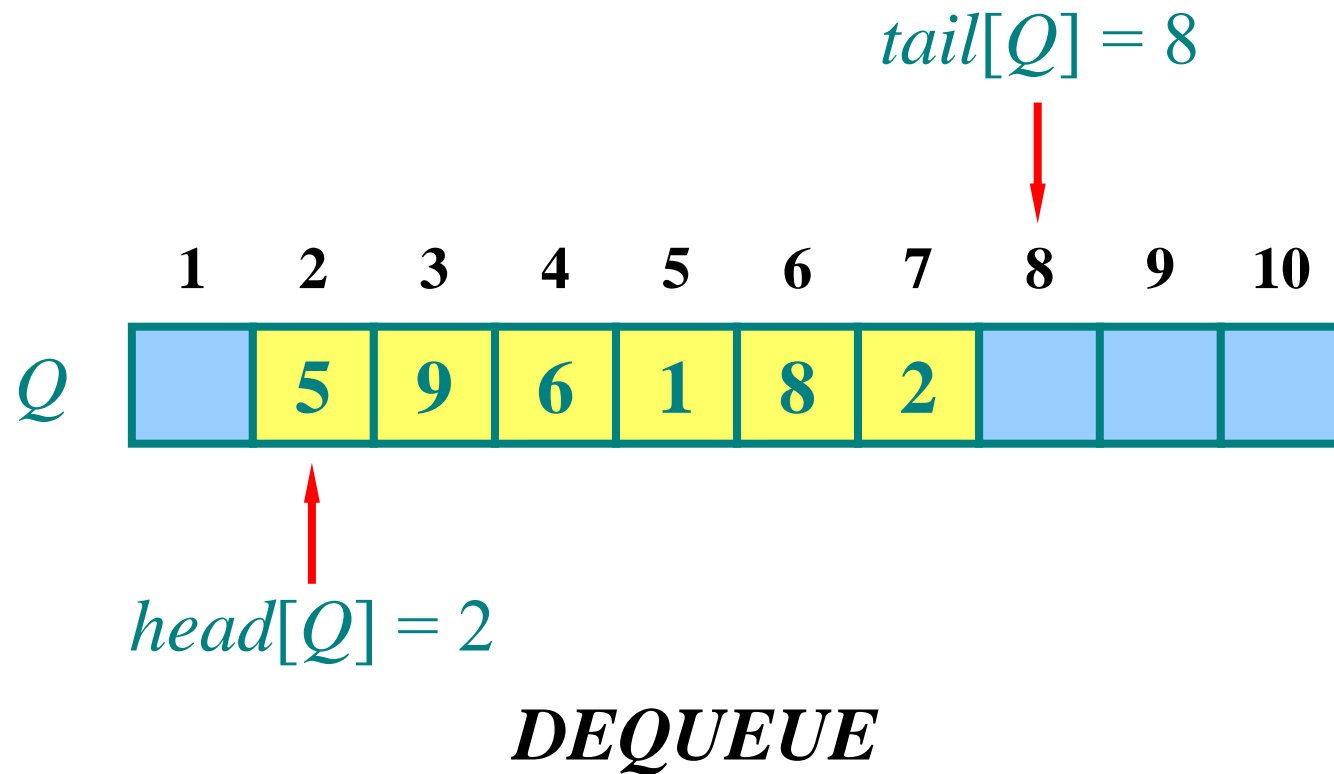


Queues

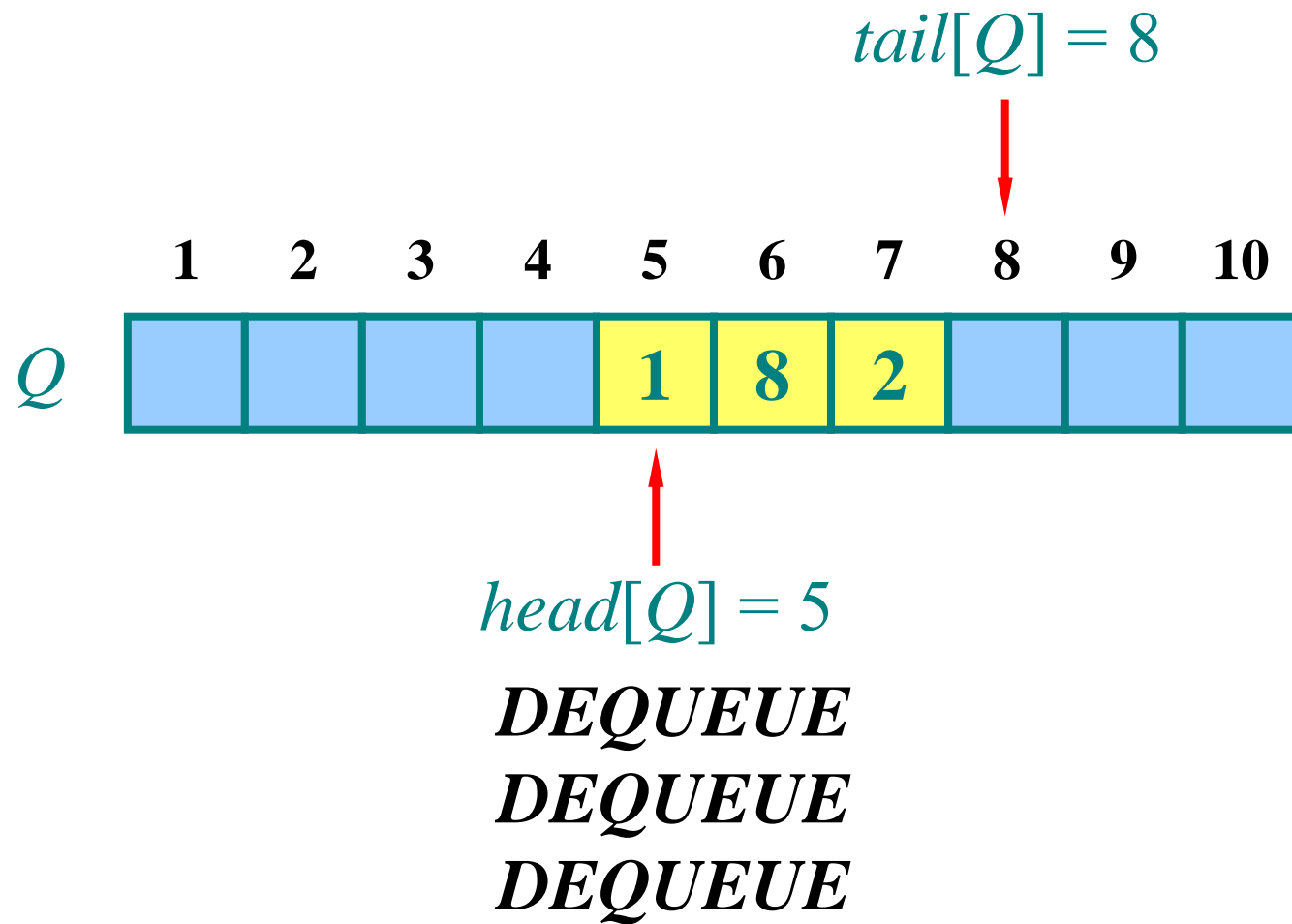


ENQUEUE 6
ENQUEUE 1
ENQUEUE 8
ENQUEUE 2

Queues

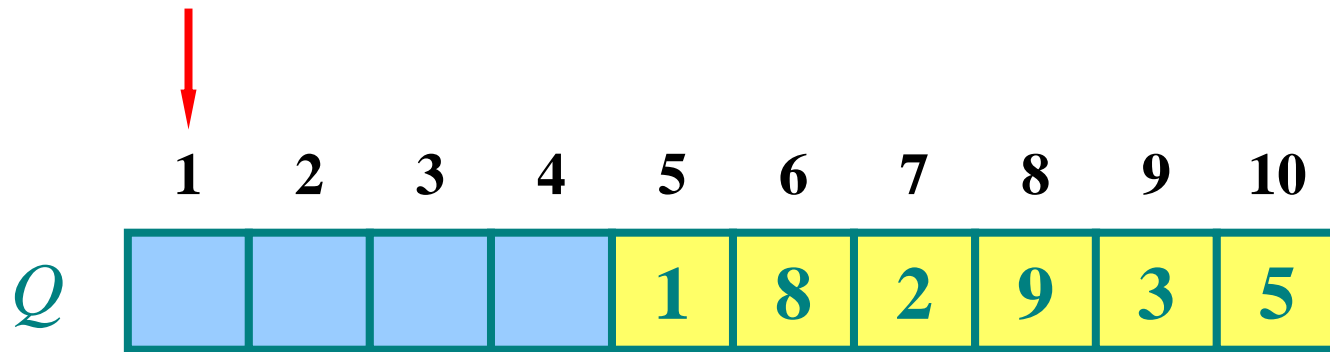


Queues



Queues

$tail[Q] = 1$



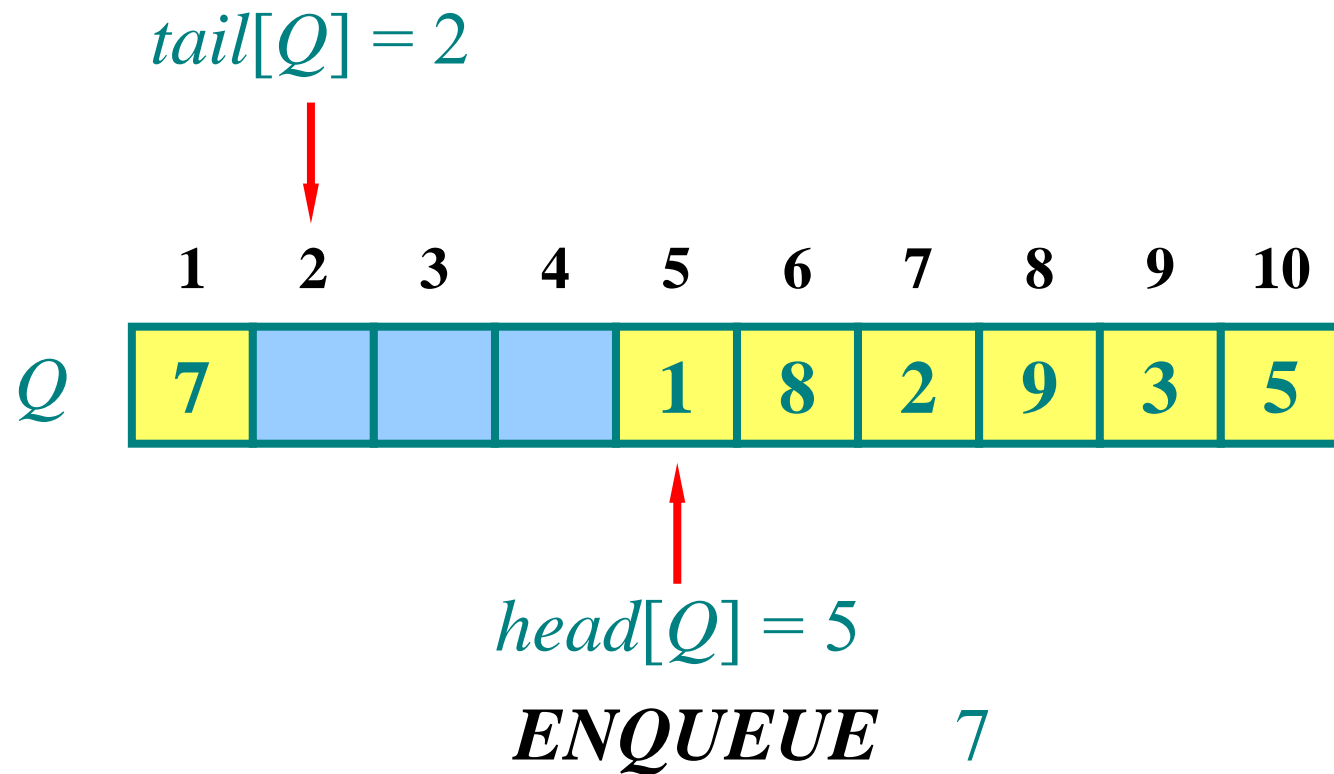
$head[Q] = 5$

ENQUEUE 9

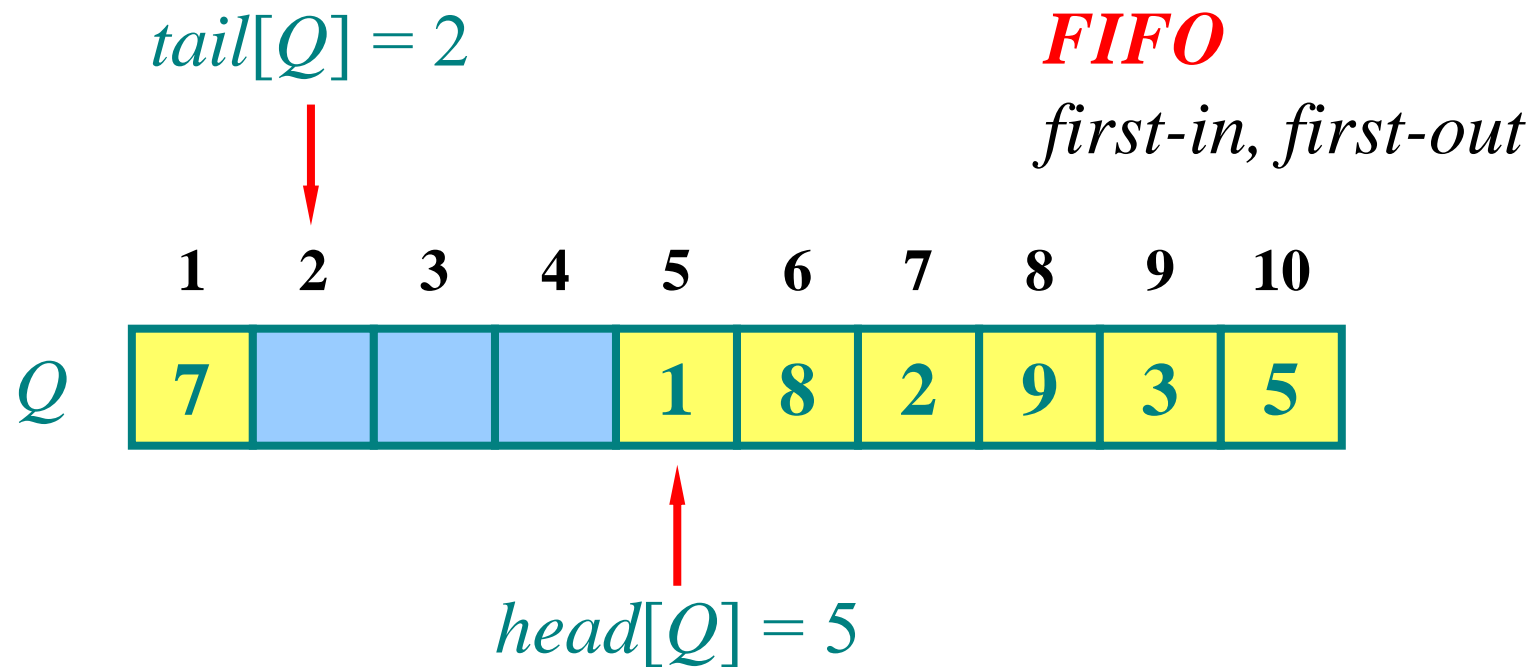
ENQUEUE 3

ENQUEUE 5

Queues



Queues



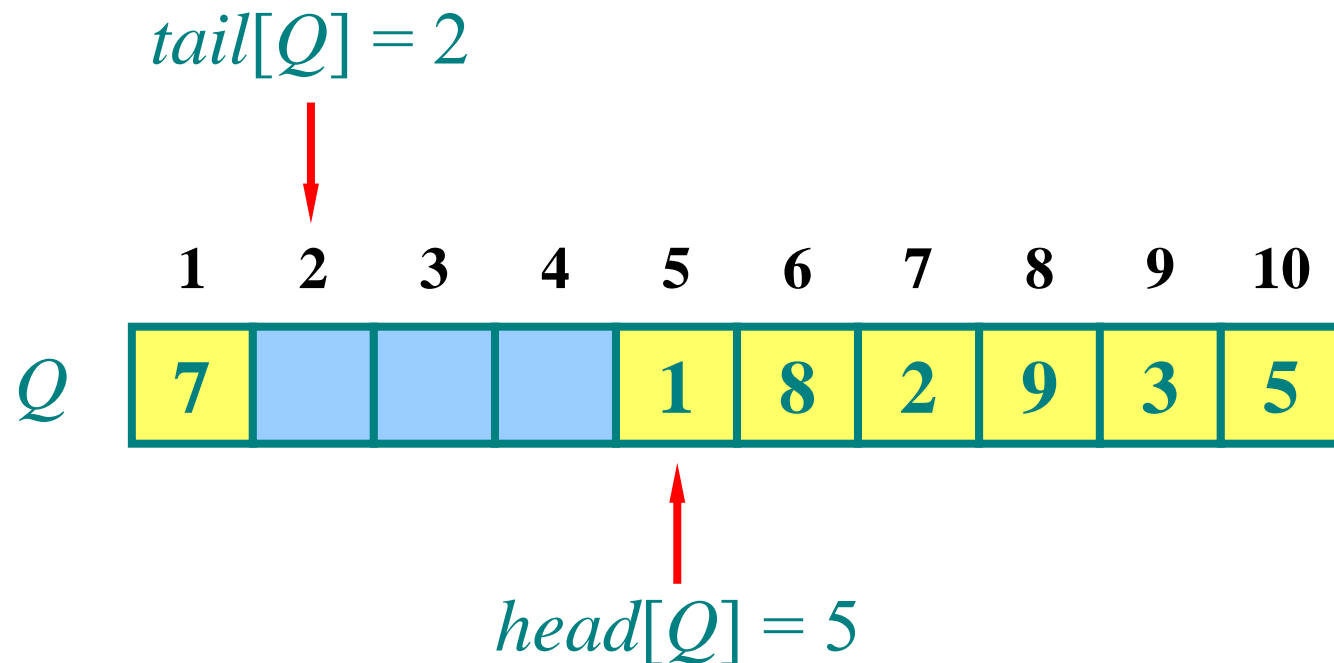
Empty

$$head[Q] = tail[Q]$$

Full

$$head[Q] = tail[Q] + 1$$

Queues



Underflow

When the queue is empty, an attempt to dequeue an element.

$$head[Q] = tail[Q]$$

Overflow

When the queue is full, an attempt to enqueue an element.

$$head[Q] = tail[Q] + 1$$

Queues

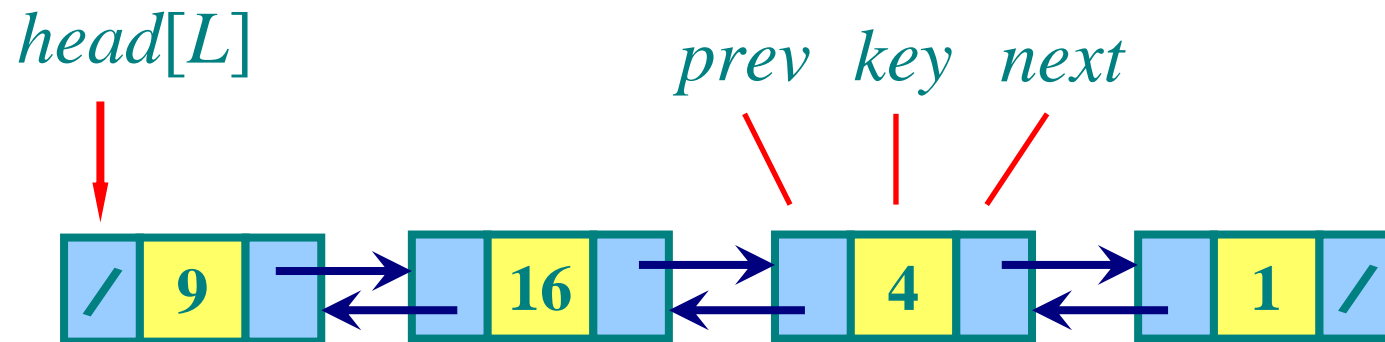
ENQUEUE(Q, x)

1. $Q[tail[Q]] \leftarrow x$
2. **if** $tail[Q] = length[Q]$
3. **then** $tail[Q] \leftarrow 1$
4. **else** $tail[Q] \leftarrow tail[Q] + 1$

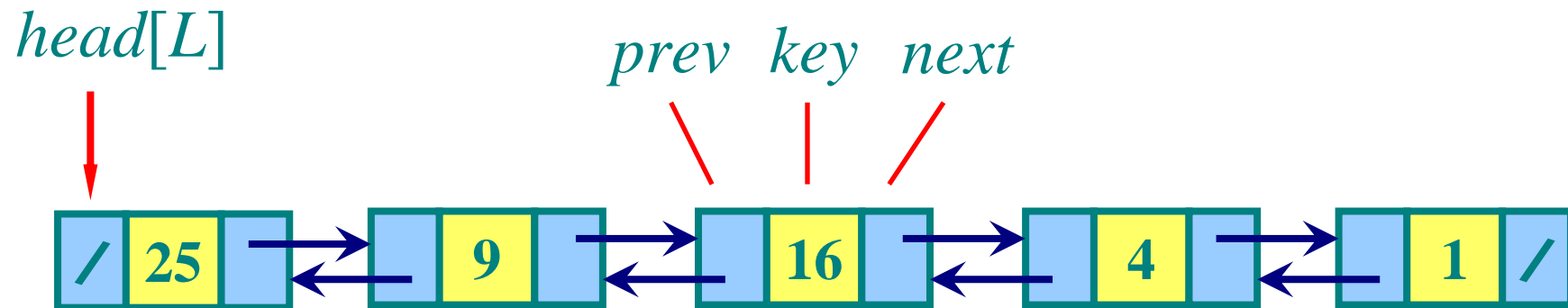
DEQUEUE(Q, x)

1. $x \leftarrow Q[head[Q]]$
2. **if** $head[Q] = length[Q]$
3. **then** $head[Q] \leftarrow 1$
4. **else** $head[Q] \leftarrow head[Q] + 1$
5. **return** x

Linked lists

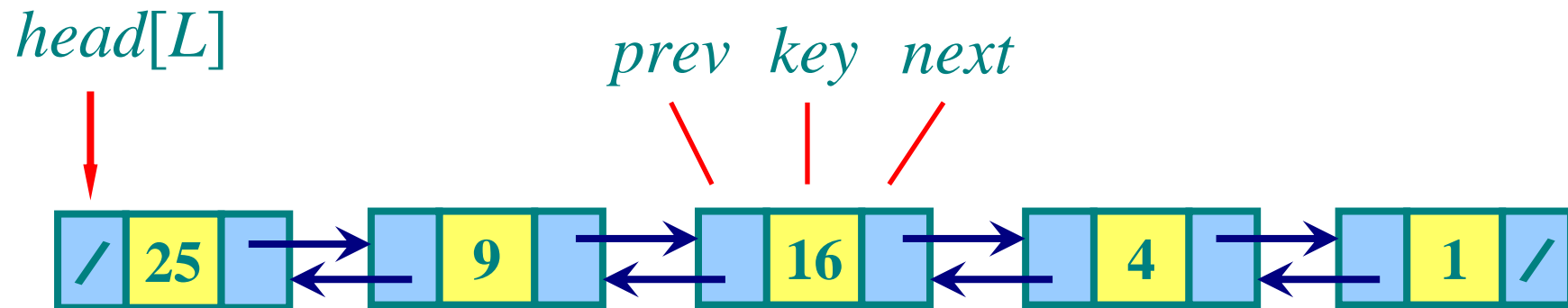


Linked lists



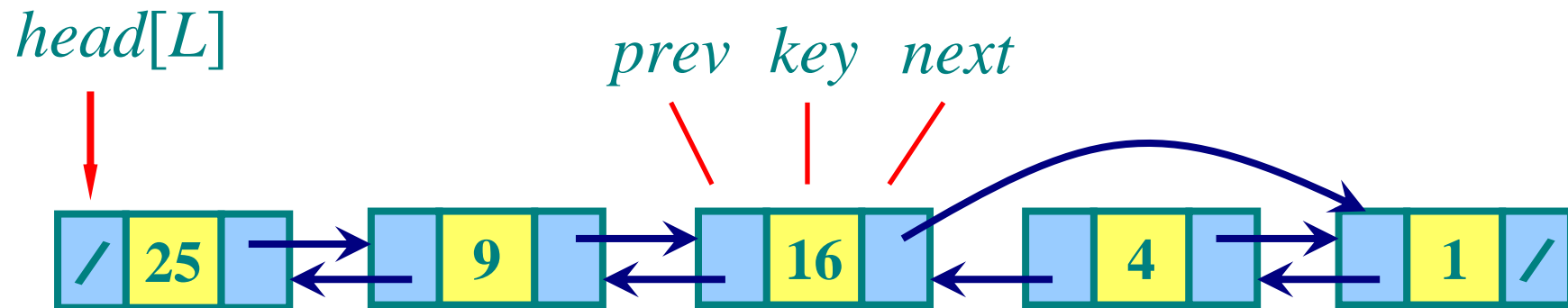
LIST-INSERT 25

Linked lists



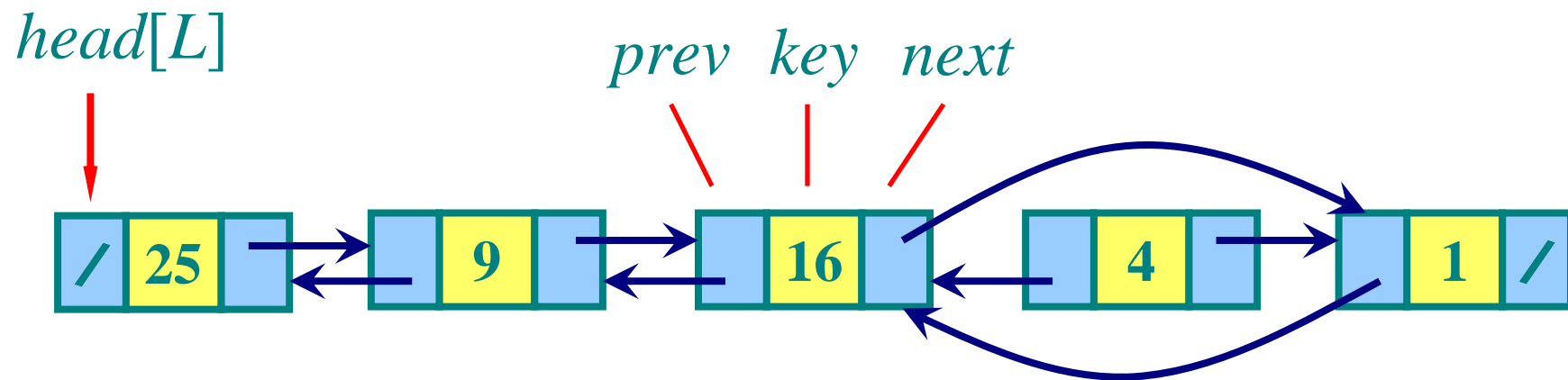
LIST-DELETE 4

Linked lists



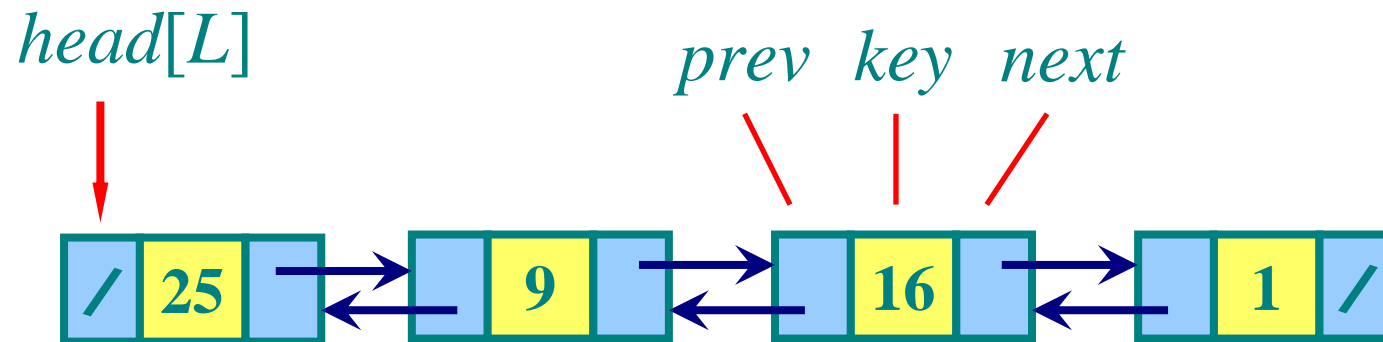
LIST-DELETE 4

Linked lists



LIST-DELETE 4

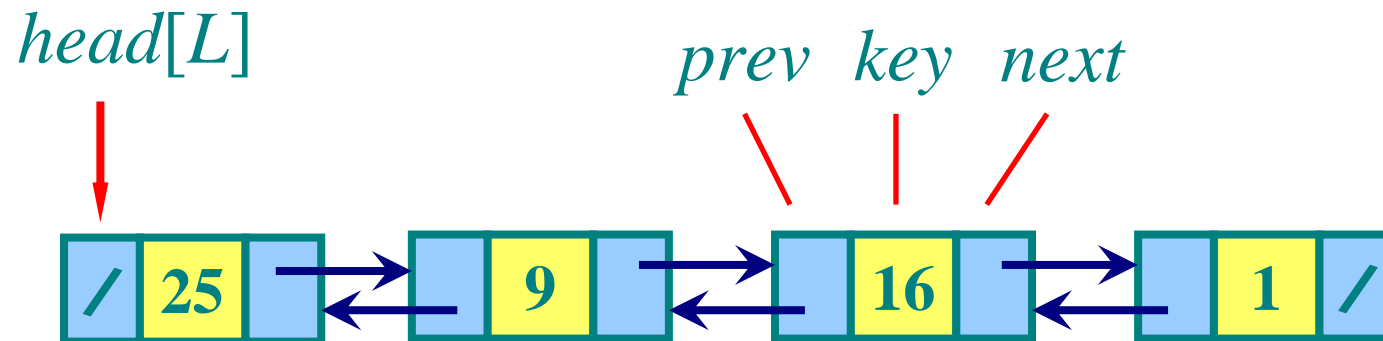
Linked lists



LIST-SEARCH(L, k)

1. $x \leftarrow \text{head}[L]$
2. **While** $x \neq \text{NIL}$ **and** $\text{key}[x] \neq k$
3. **do** $x \leftarrow \text{next}[x]$
4. **return** x

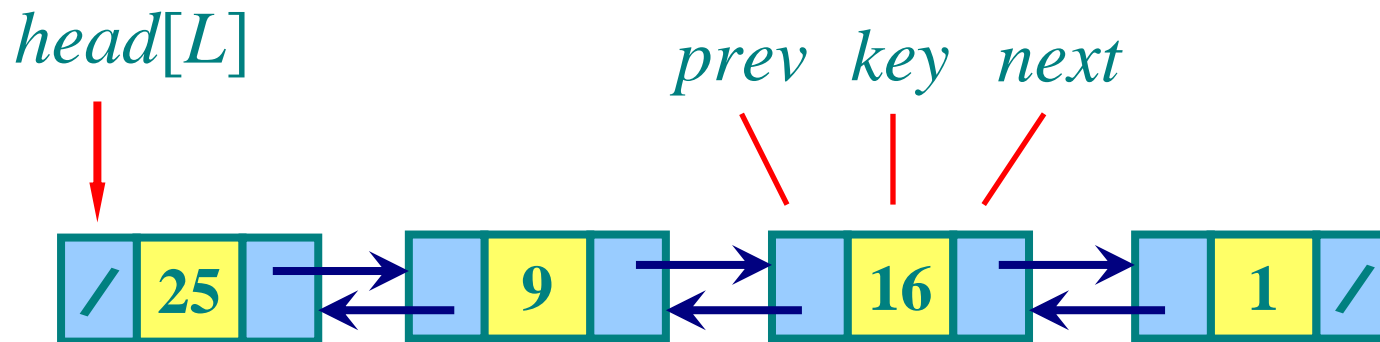
Linked lists



LIST-INSERT(L, x)

1. $next[x] \leftarrow head[L]$
2. **if** $head[L] \neq \text{NIL}$
3. **then** $prev[head[L]] \leftarrow x$
4. $head[L] \leftarrow x$
5. $prev[x] \leftarrow \text{NIL}$

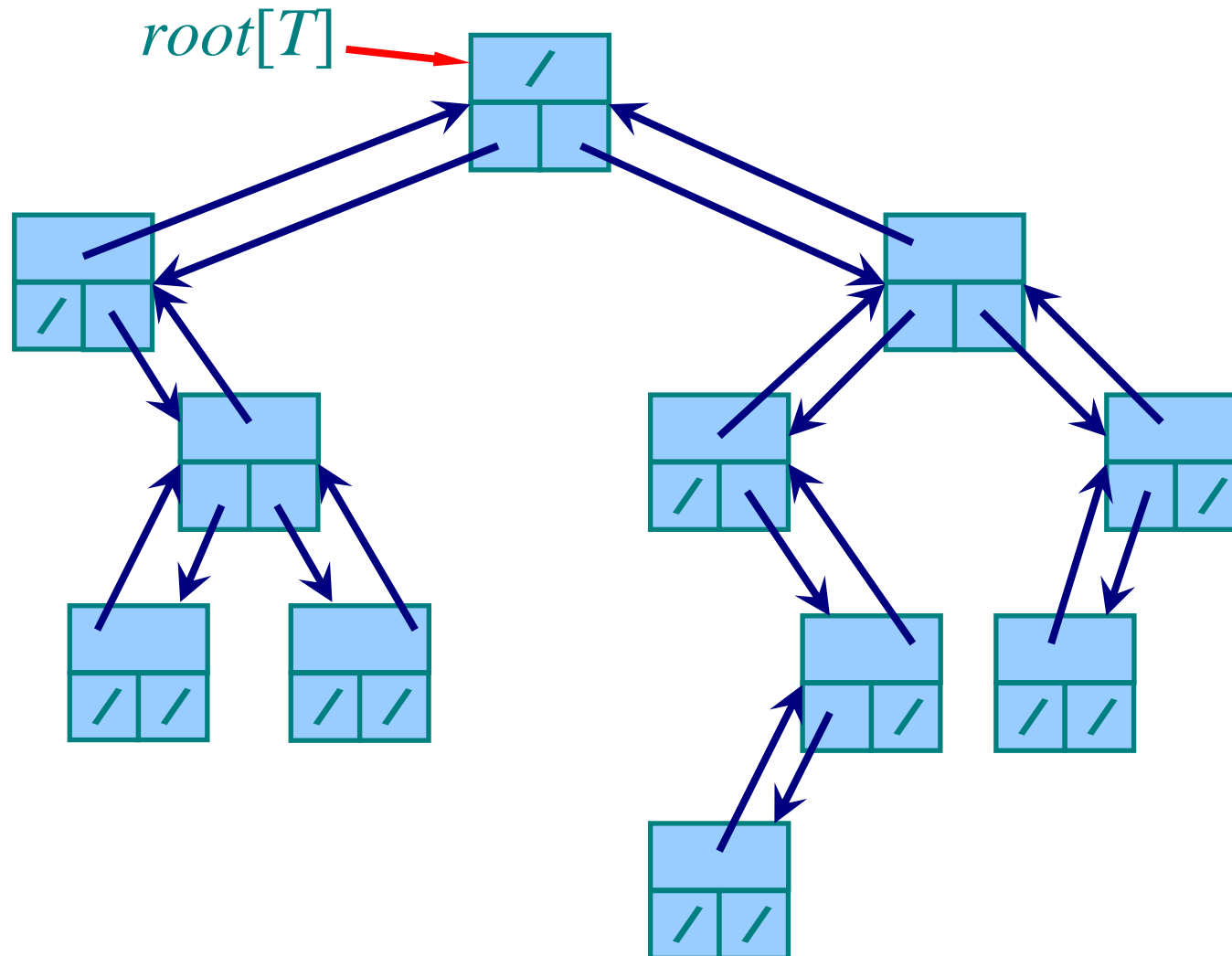
Linked lists



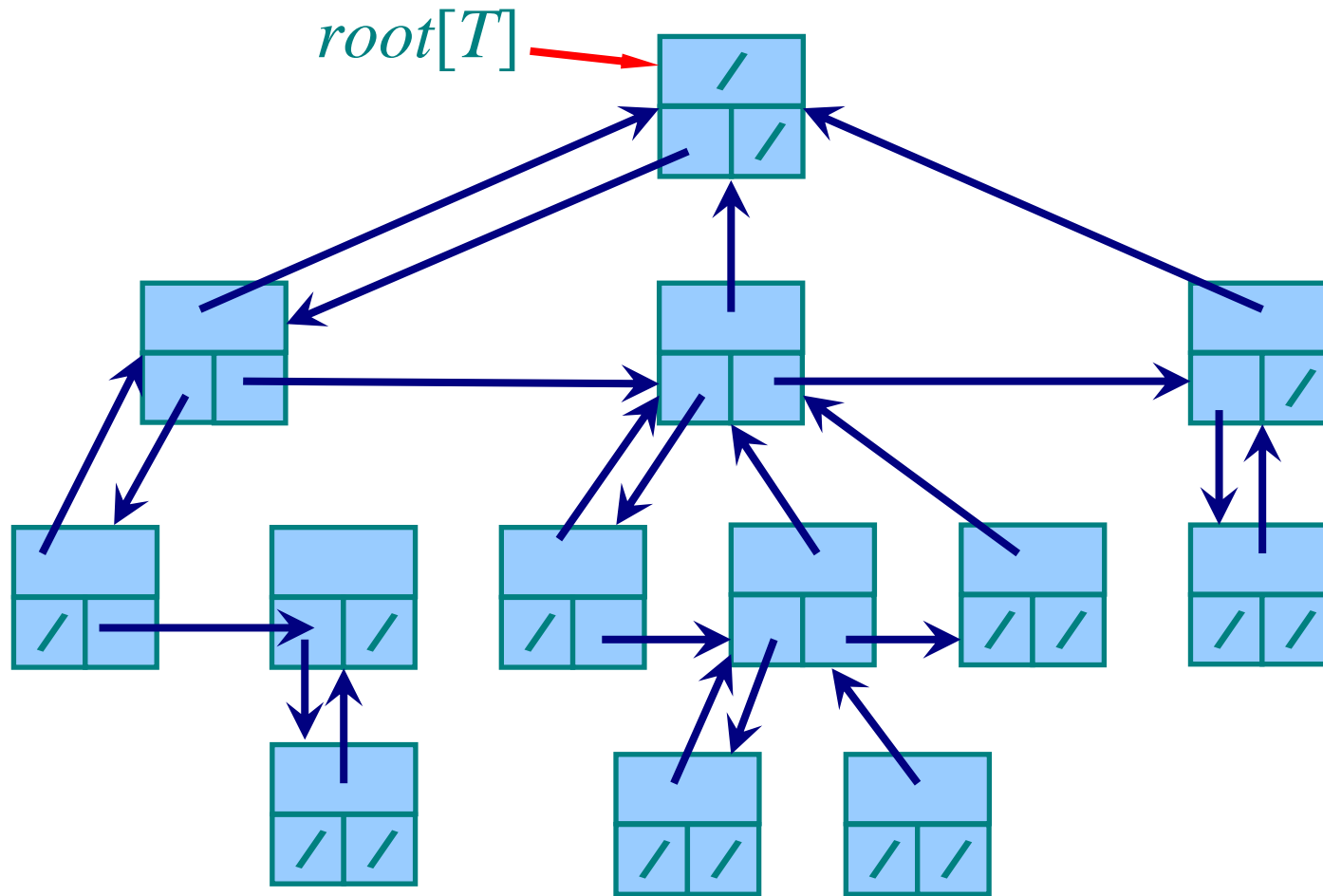
LIST-DELETE(L, x)

1. **if** $prev[x] \neq \text{NIL}$
2. **then** $next[prev[x]] \leftarrow next[x]$
3. **else** $head[L] \leftarrow next[x]$
4. **if** $next[x] \neq \text{NIL}$
5. **then** $prev[next[x]] \leftarrow prev[x]$

Rooted trees



Rooted trees



Postfix

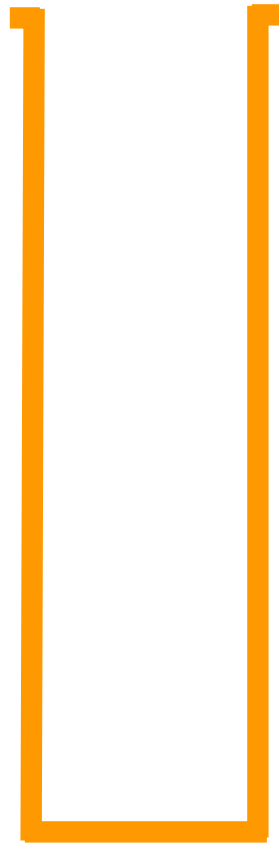
Infix

$$9 + 3 * 7 + (4 * 5 + 6) * 2$$

Postfix (reverse Polish notation)

$$9\ 3\ 7\ *\ +\ 4\ 5\ *\ 6\ +\ 2\ *\ +$$

Postfix expression



Stack

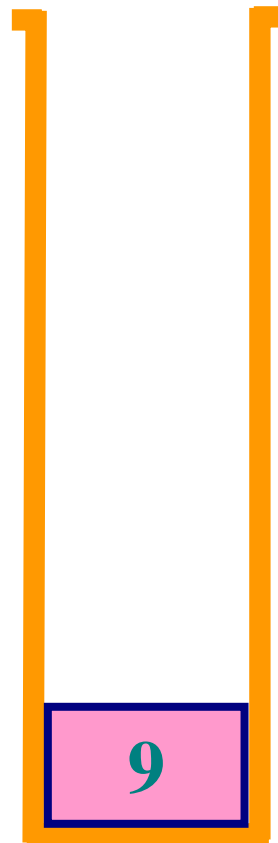
Infix

$9 + 3 * 7 + (4 * 5 + 6) * 2$

Input

9 3 7 * + 4 5 * 6 + 2 * +

Postfix expression



Infix

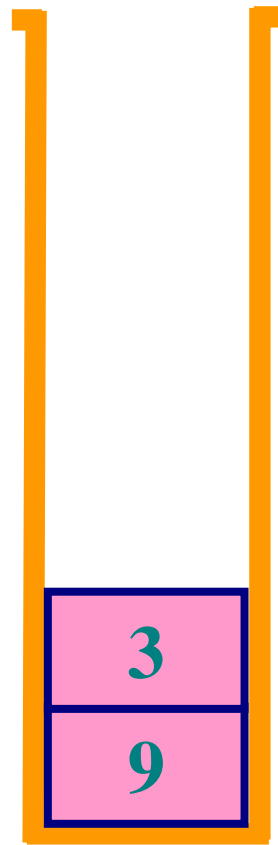
$9 + 3 * 7 + (4 * 5 + 6) * 2$

Input

9 3 7 * + 4 5 * 6 + 2 * +



Postfix expression



Stack

Infix

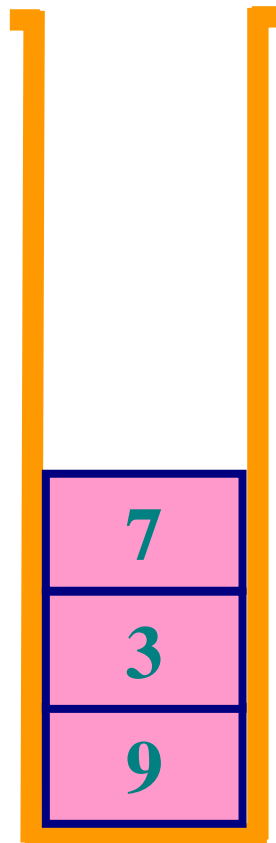
$9 + 3 * 7 + (4 * 5 + 6) * 2$

Input

9 3 7 * + 4 5 * 6 + 2 * +



Postfix expression



Stack

Infix

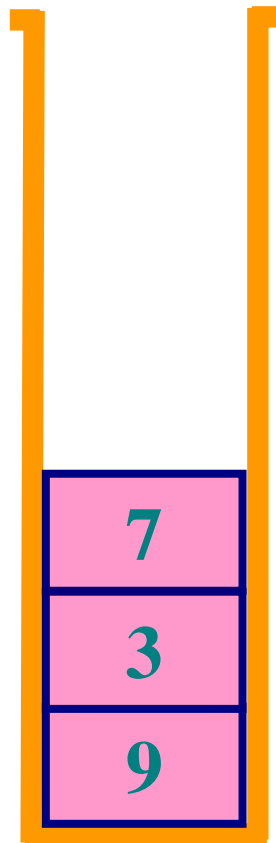
$9 + 3 * 7 + (4 * 5 + 6) * 2$

Input

9 3 7 * + 4 5 * 6 + 2 * +



Postfix expression



Stack

Infix

$9 + 3 * 7 + (4 * 5 + 6) * 2$

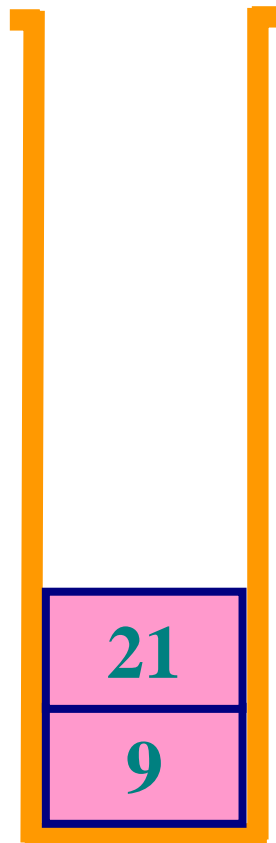
Input

9 3 7 * + 4 5 * 6 + 2 * +



3 and 7 are popped and $3 * 7 = 21$
is pushed

Postfix expression



Stack

Infix

$9 + 3 * 7 + (4 * 5 + 6) * 2$

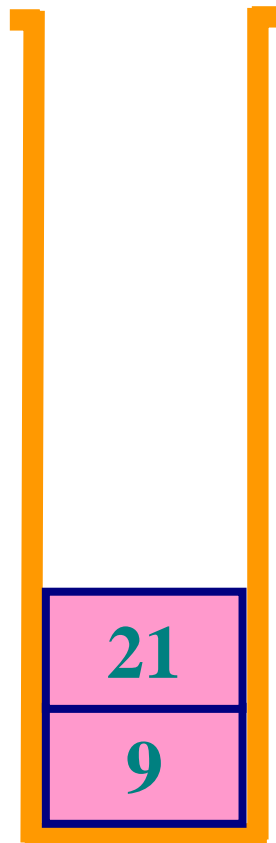
Input

9 3 7 * + 4 5 * 6 + 2 * +



3 and 7 are popped and $3 * 7 = 21$
is pushed

Postfix expression



Stack

Infix

$9 + 3 * 7 + (4 * 5 + 6) * 2$

Input

9 3 7 * + 4 5 * 6 + 2 * +



9 and 21 are popped and $21 + 9 = 30$ is pushed

Postfix expression



Infix

$9 + 3 * 7 + (4 * 5 + 6) * 2$

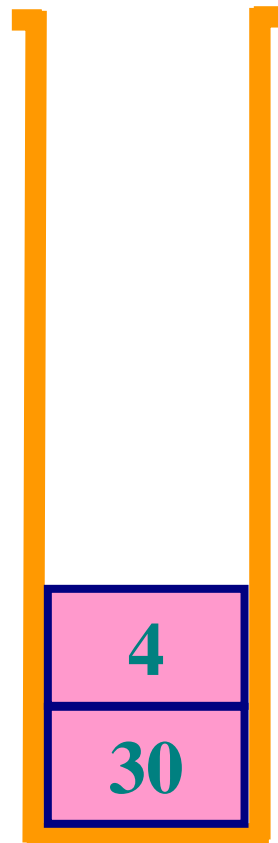
Input

9 3 7 * + 4 5 * 6 + 2 * +



9 and 21 are popped and $21 + 9 = 30$
is pushed

Postfix expression



Stack

Infix

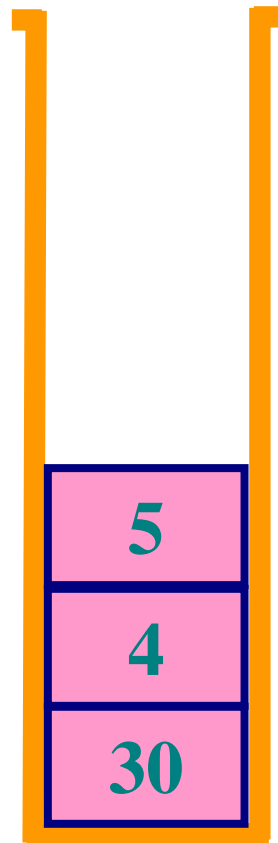
$9 + 3 * 7 + (4 * 5 + 6) * 2$

Input

9 3 7 * + 4 5 * 6 + 2 * +



Postfix expression



Stack

Infix

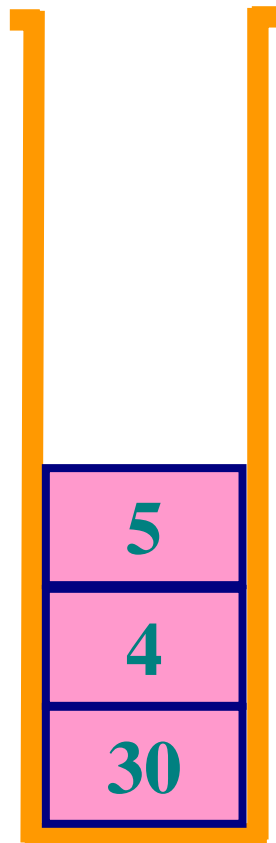
$9 + 3 * 7 + (4 * 5 + 6) * 2$

Input

9 3 7 * + 4 5 * 6 + 2 * +



Postfix expression



Stack

Infix

$9 + 3 * 7 + (4 * 5 + 6) * 2$

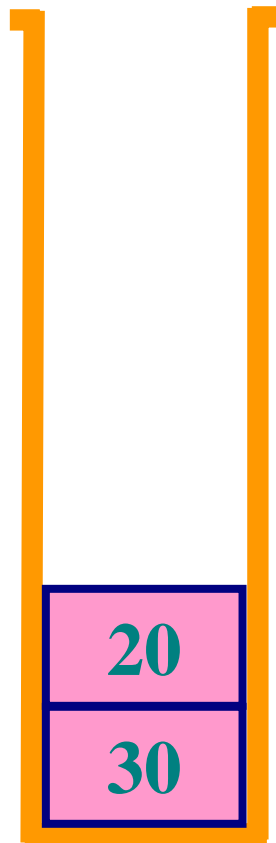
Input

9 3 7 * + 4 5 * 6 + 2 * +



4 and 5 are popped and $4 * 5 = 20$ is pushed

Postfix expression



Stack

Infix

$9 + 3 * 7 + (4 * 5 + 6) * 2$

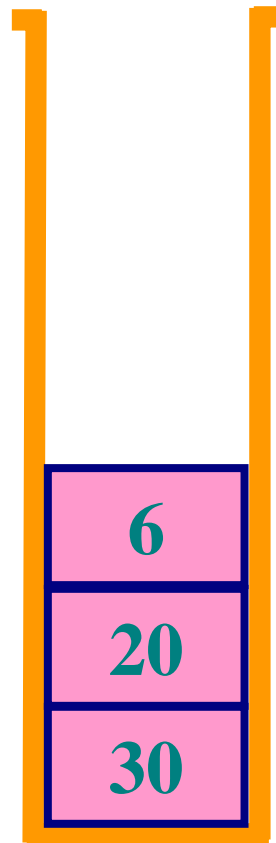
Input

9 3 7 * + 4 5 * 6 + 2 * +



4 and 5 are popped and $4 * 5 = 20$ is pushed

Postfix expression



Stack

Infix

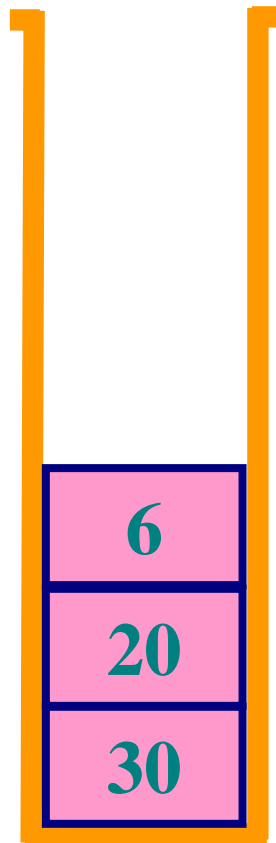
$9 + 3 * 7 + (4 * 5 + 6) * 2$

Input

9 3 7 * + 4 5 * 6 + 2 * +



Postfix expression



Stack

Infix

$9 + 3 * 7 + (4 * 5 + 6) * 2$

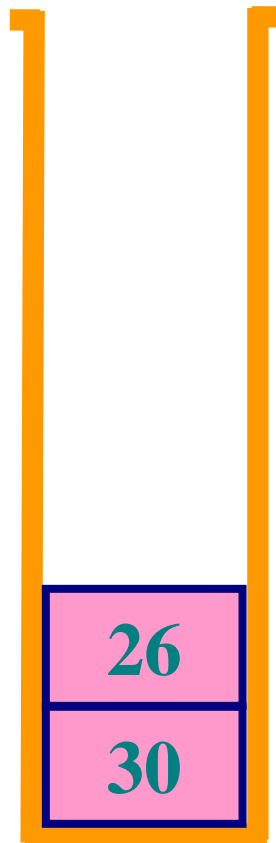
Input

9 3 7 * + 4 5 * 6 + 2 * +



6 and 20 are popped and $6 + 20 = 26$ is pushed

Postfix expression



Stack

Infix

$9 + 3 * 7 + (4 * 5 + 6) * 2$

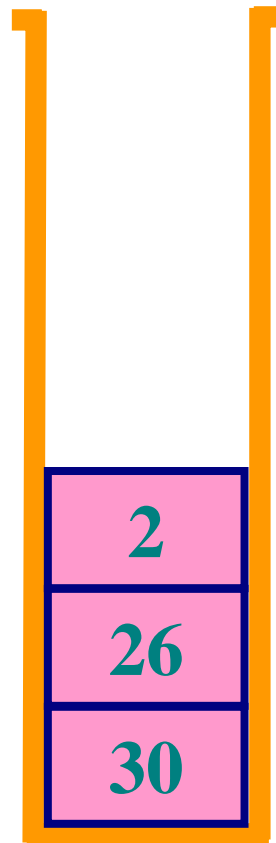
Input

9 3 7 * + 4 5 * 6 + 2 * +



6 and 20 are popped and $6 + 20 = 26$
is pushed

Postfix expression



Stack

Infix

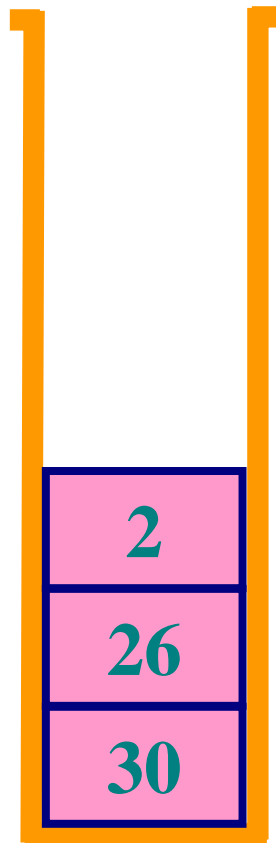
$9 + 3 * 7 + (4 * 5 + 6) * 2$

Input

9 3 7 * + 4 5 * 6 + 2 * +



Postfix expression



Stack

Infix

$9 + 3 * 7 + (4 * 5 + 6) * 2$

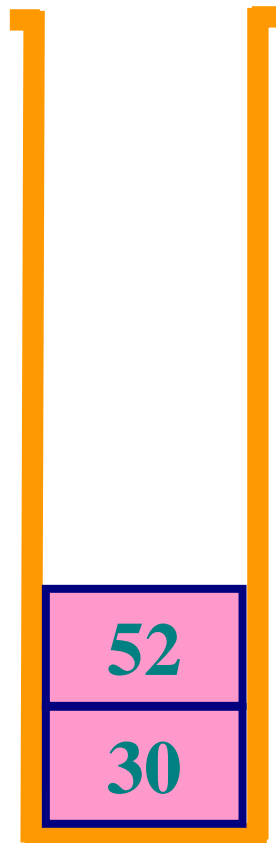
Input

9 3 7 * + 4 5 * 6 + 2 * +



2 and 26 are popped and $2 * 26 = 52$
is pushed

Postfix expression



Stack

Infix

$9 + 3 * 7 + (4 * 5 + 6) * 2$

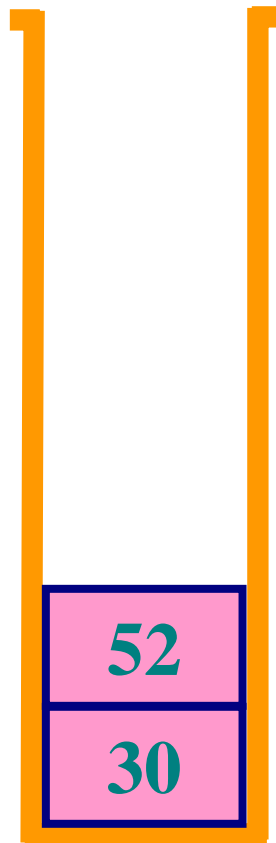
Input

9 3 7 * + 4 5 * 6 + 2 * +



2 and 26 are popped and $2 * 26 = 52$
is pushed

Postfix expression



Stack

Infix

$9 + 3 * 7 + (4 * 5 + 6) * 2$

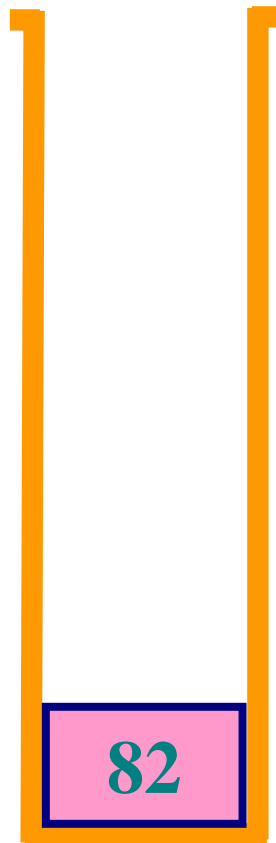
Input

9 3 7 * + 4 5 * 6 + 2 * +



52 and 30 are popped and $52 + 30 = 82$ is pushed

Postfix expression



Infix

$9 + 3 * 7 + (4 * 5 + 6) * 2$

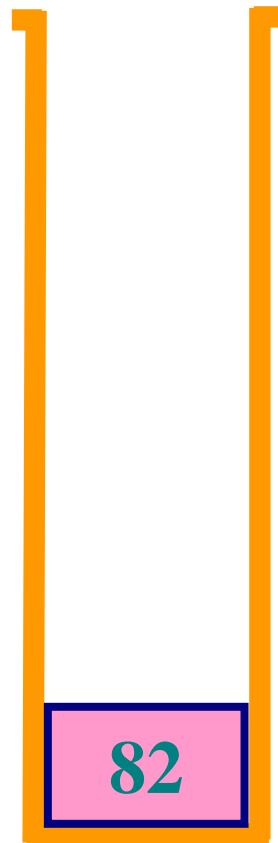
Input

9 3 7 * + 4 5 * 6 + 2 * +



52 and 30 are popped and $52 + 30 = 82$
is pushed

Postfix expression



Infix

$9 + 3 * 7 + (4 * 5 + 6) * 2$

done

Input

9 3 7 * + 4 5 * 6 + 2 * +



82 are popped and we get the answer

Stack

Conversion rules

- ❑ Operands are immediately placed onto the output and operators are placed onto the stack .
- ❑ When the operator on the top of the stack that has higher priority than the input operator, it is popped.
- ❑ If we see a right parenthesis, then we pop the stack, writing symbols until we encounter a left parenthesis.
- ❑ If we read the end of input, we pop the stack until it is empty.

Infix to postfix conversion



Stack

Input

$9 + 3 * 7 + (4 * 5 + 6) * 2$

Output

Infix to postfix conversion



Stack

Input

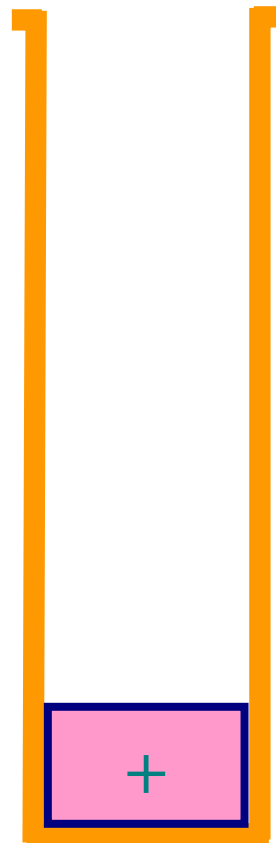
$9 + 3 * 7 + (4 * 5 + 6) * 2$



Output

9

Infix to postfix conversion



Stack

Input

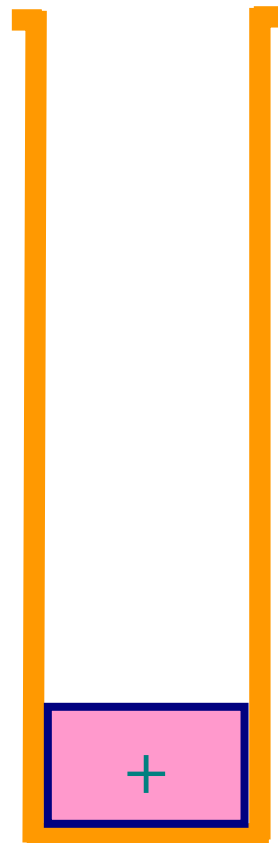
$9 + 3 * 7 + (4 * 5 + 6) * 2$



Output

9

Infix to postfix conversion



Stack

Input

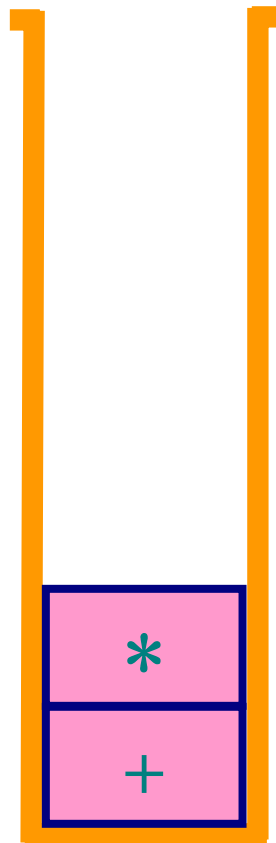
9 + 3 * 7 + (4 * 5 + 6) * 2



Output

9 3

Infix to postfix conversion



Stack

Input

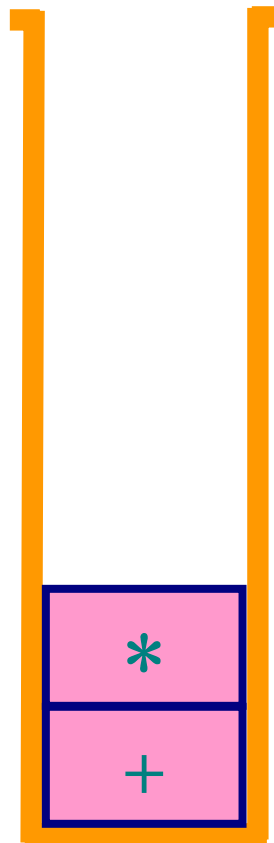
$9 + 3 * 7 + (4 * 5 + 6) * 2$



Output

9 3

Infix to postfix conversion



Stack

Input

$9 + 3 * 7 + (4 * 5 + 6) * 2$



Output

$9 \ 3 \ 7$

Infix to postfix conversion

$+$ has lower priority than $*$

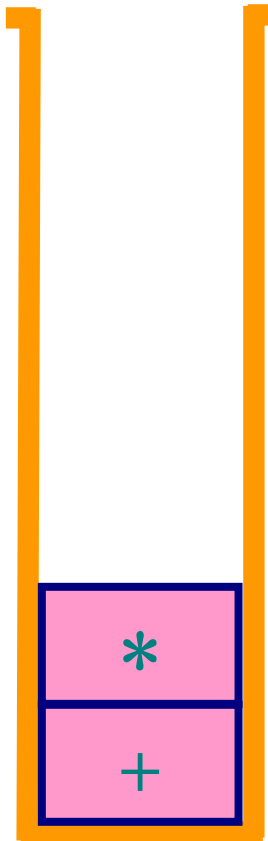
Input

9 + 3 * 7 + (4 * 5 + 6) * 2



Output

9 3 7 *



Stack

Infix to postfix conversion

Pop the other +

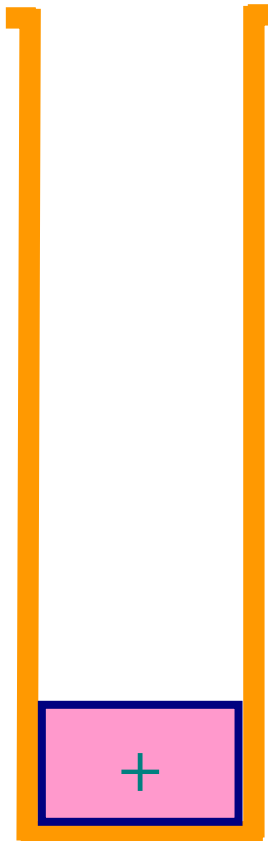
Input

9 + 3 * 7 + (4 * 5 + 6) * 2



Output

9 3 7 * +



Stack

Infix to postfix conversion

Push the input $+$ onto stack

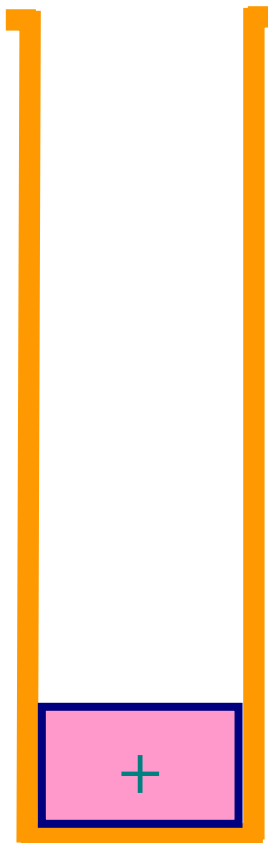
Input

9 + 3 * 7 + (4 * 5 + 6) * 2



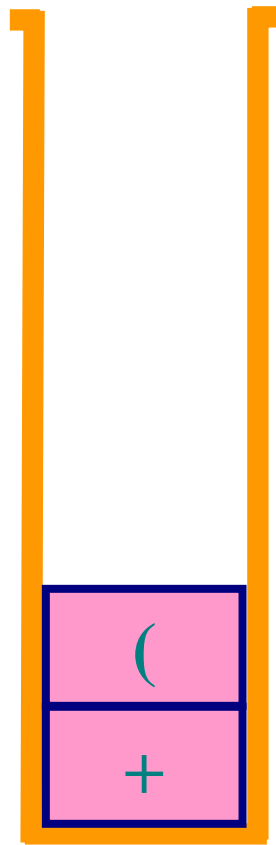
Output

9 3 7 * +



Stack

Infix to postfix conversion



Stack

Input

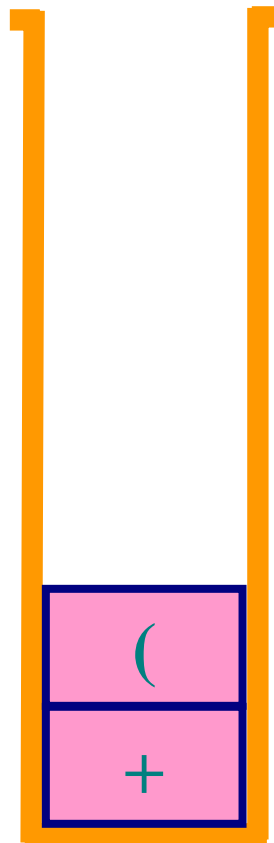
9 + 3 * 7 + (4 * 5 + 6) * 2



Output

9 3 7 * +

Infix to postfix conversion



Stack

Input

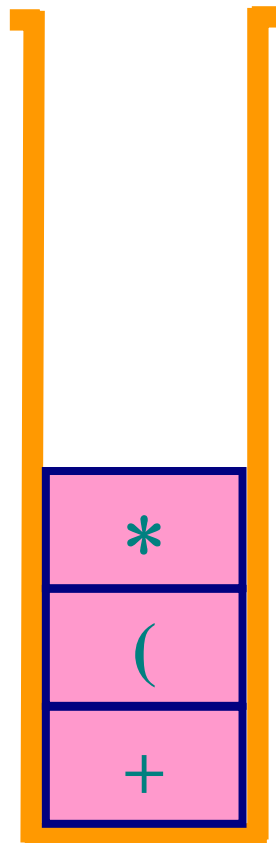
9 + 3 * 7 + (4 * 5 + 6) * 2



Output

9 3 7 * + 4

Infix to postfix conversion



Stack

Input

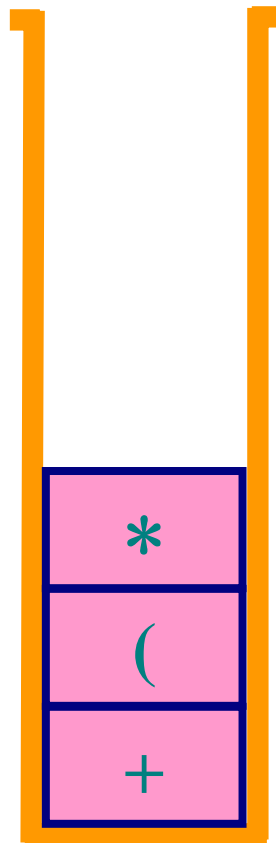
$9 + 3 * 7 + (4 * 5 + 6) * 2$



Output

$9 \ 3 \ 7 \ * \ + \ 4$

Infix to postfix conversion



Stack

Input

$9 + 3 * 7 + (4 * 5 + 6) * 2$



Output

$9 \ 3 \ 7 \ * \ + \ 4 \ 5$

Infix to postfix conversion

$+$ has lower priority than $*$

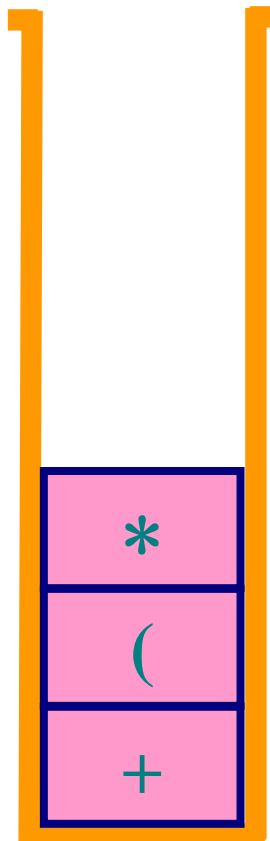
Input

$9 + 3 * 7 + (4 * 5 + 6) * 2$



Output

$9\ 3\ 7\ *\ +\ 4\ 5\ *$



Stack

Infix to postfix conversion

Push the input $+$ onto stack

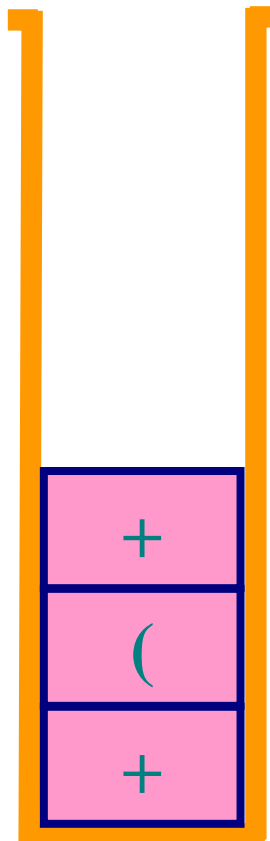
Input

$9 + 3 * 7 + (4 * 5 + 6) * 2$



Output

$9\ 3\ 7\ * +\ 4\ 5\ *$



Stack

Infix to postfix conversion

Push the input $+$ onto stack

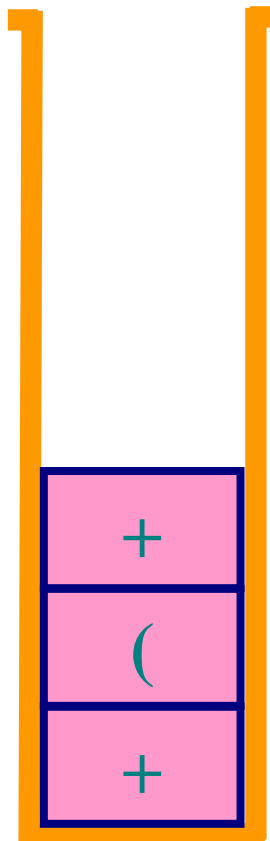
Input

$9 + 3 * 7 + (4 * 5 + 6) * 2$



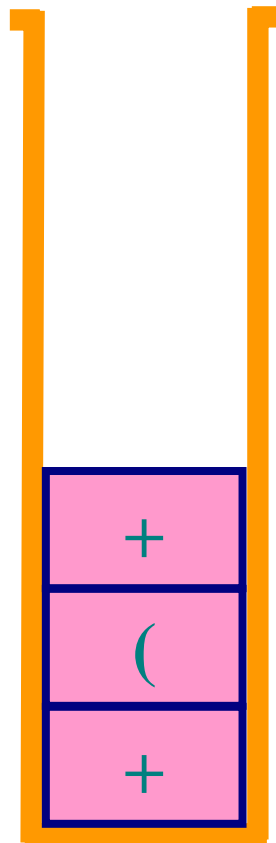
Output

$9\ 3\ 7\ *\ +\ 4\ 5\ *\ 6$



Stack

Infix to postfix conversion



Input

9 + 3 * 7 + (4 * 5 + 6) * 2

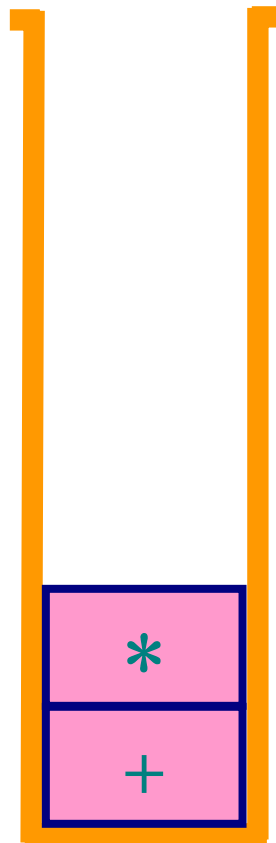


Pop the stack until we encounter left parenthesis

Output

9 3 7 * + 4 5 * 6 +

Infix to postfix conversion



Stack

Input

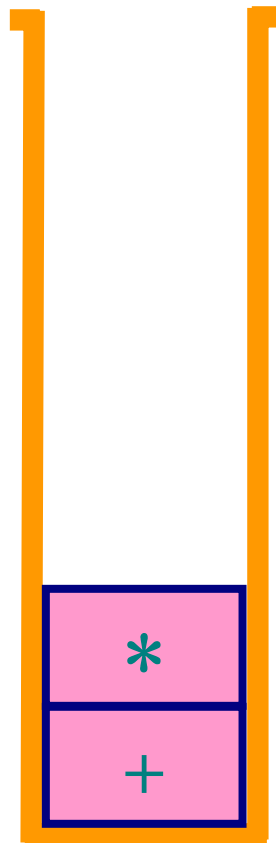
$9 + 3 * 7 + (4 * 5 + 6) * 2$



Output

$9\ 3\ 7\ *\ +\ 4\ 5\ *\ 6\ +$

Infix to postfix conversion



Stack

Input

$9 + 3 * 7 + (4 * 5 + 6) * 2$



Output

$9\ 3\ 7\ * +\ 4\ 5\ * 6 + 2$

Infix to postfix conversion

Pop the stack until it is empty

Input

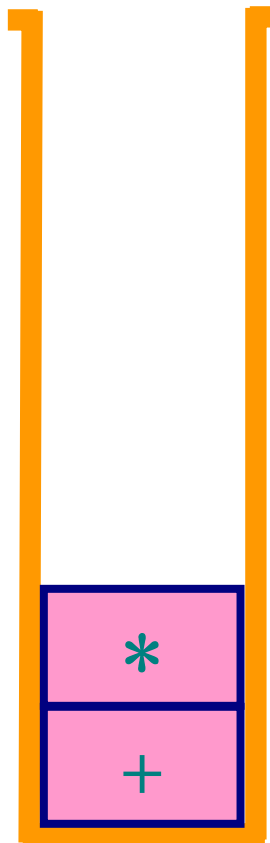
9 + 3 * 7 + (4 * 5 + 6) * 2



Output

9 3 7 * + 4 5 * 6 + 2 * +

done



Stack

Any question?



Xiaoqing Zheng
Fudan University