Stacks
- Application 1

Stacks
- Application 1

Matching Paranthesis

For every open paranthesis '(', there exists a following ')'.

Stacks
- Application 1

Matching Paranthesis

For every open paranthesis '(', there exists a following ')'.

Examples

Stacks
- Application 1

Matching Paranthesis

For every open paranthesis '(', there exists a following ')'.

Examples

$$(\; (\;) \;)$$

Stacks
- Application 1

Matching Paranthesis

For every open paranthesis '(', there exists a following ')'.

Examples

(( ))

(( )) (( ))

Stacks
- Application 1

Matching Paranthesis

For every open paranthesis '(', there exists a following ')'.

Examples

(( ))

(( )) (( ))

) (

Stacks
- Application 1

Matching Paranthesis

For every open paranthesis '(', there exists a following ')'.

Examples

(( ))

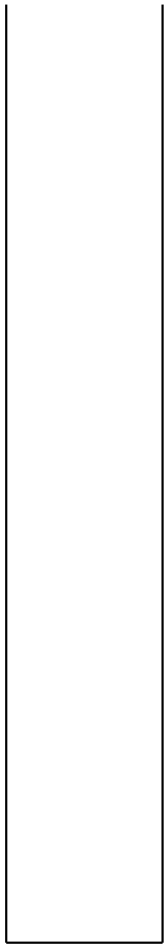
(( )) (( ))

) (

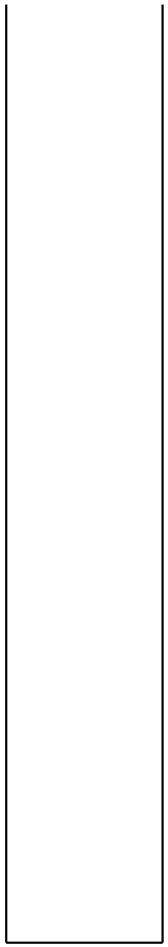
( ) (

Stacks
- Paranthesis Matching

Stacks
- Paranthesis Matching

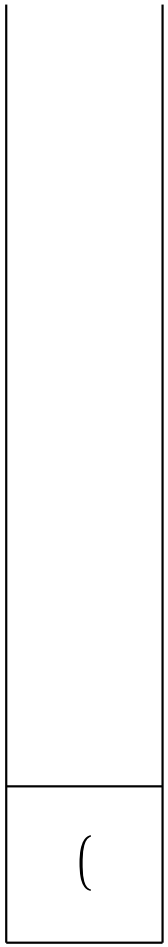


Stacks
- Paranthesis Matching



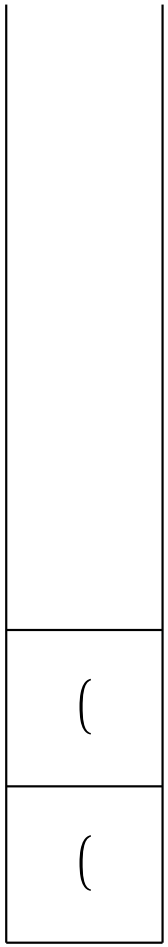
(( ))

Stacks
- Paranthesis Matching



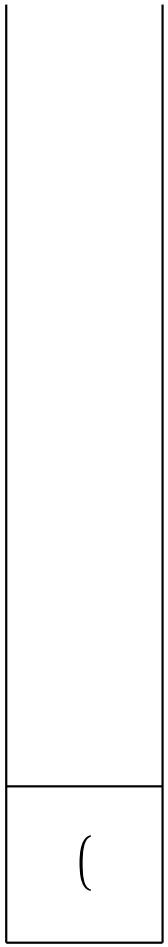
(( ))

Stacks
- Paranthesis Matching



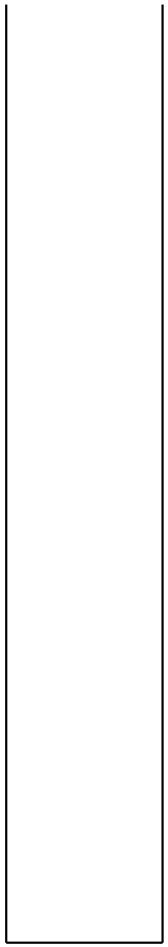
(( ))

Stacks
- Paranthesis Matching



(( ))

Stacks
- Paranthesis Matching

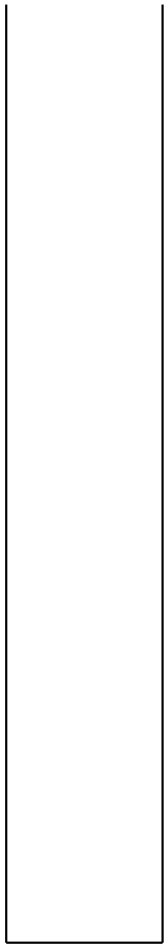


(( ))

Matching!

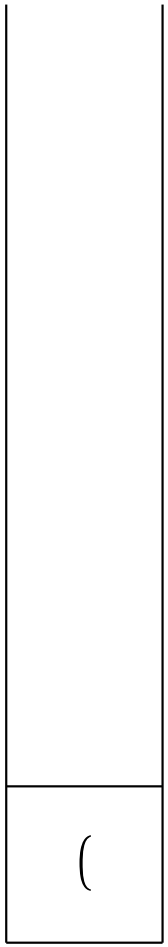
(i) Input read;
(ii) stack empty.

Stacks
- Paranthesis Matching



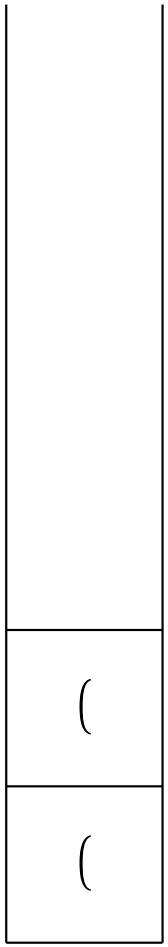
(( )) (( ))

Stacks
- Paranthesis Matching



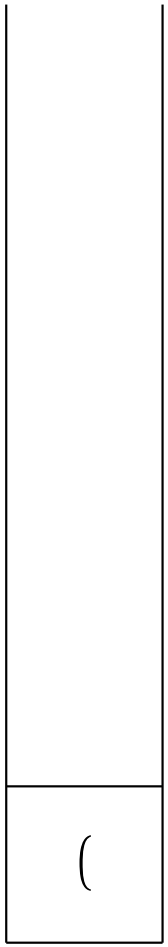
(( )) (( ))

Stacks
- Paranthesis Matching



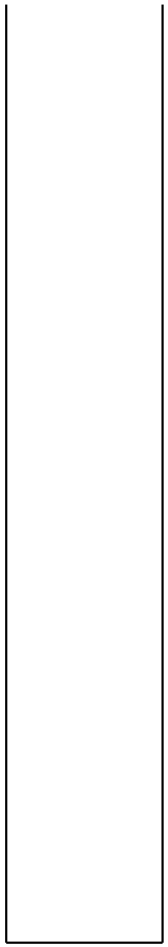
(( )) (( ))

Stacks
- Paranthesis Matching



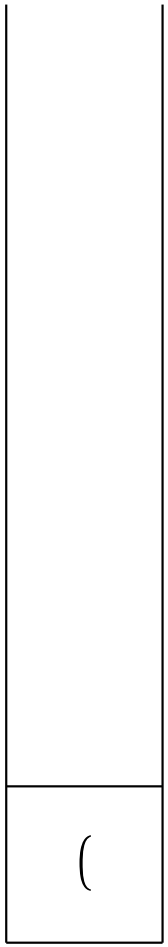
(( )) (( ))

Stacks
- Paranthesis Matching



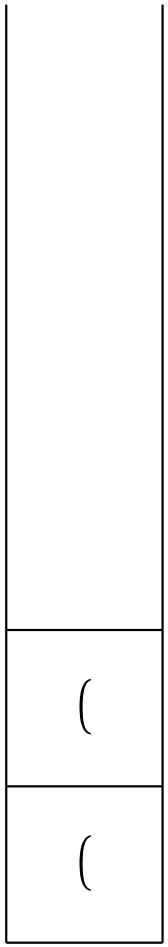
(( )) (( ))

Stacks
- Paranthesis Matching



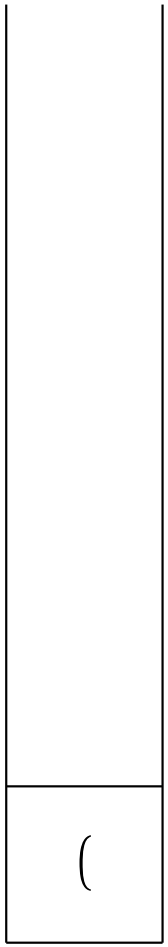
(( )) (( ))

Stacks
- Paranthesis Matching



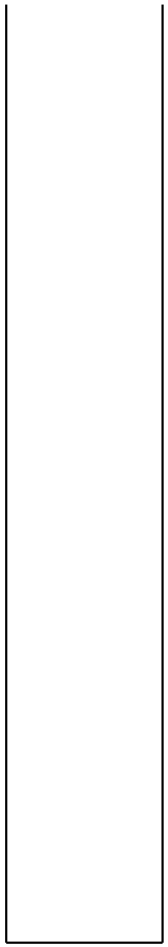
(( )) (( ))

Stacks
- Paranthesis Matching



(( )) (( ))

Stacks
- Paranthesis Matching

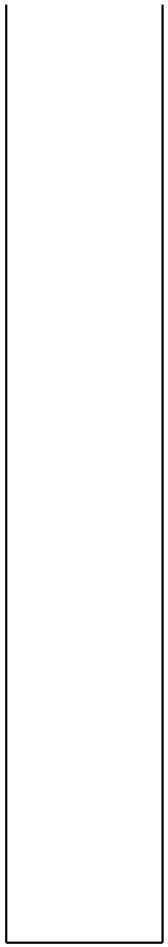


(( )) (( ))

Matching!

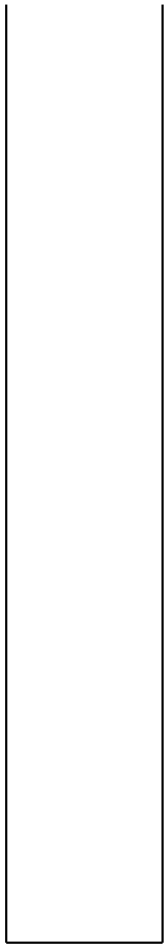
(i) Input read;
(ii) stack empty.

Stacks
- Paranthesis Matching



) (

Stacks
- Paranthesis Matching

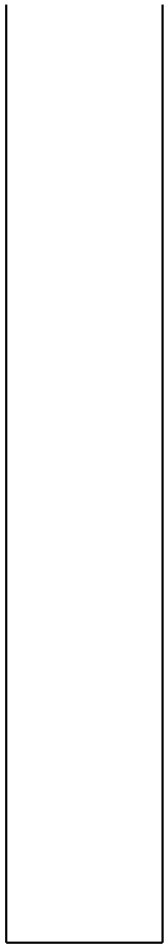


) (

Not Matching!

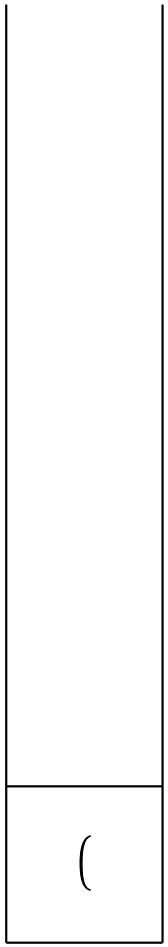
(i) Input not read;
(ii) stack empty.

Stacks
- Paranthesis Matching



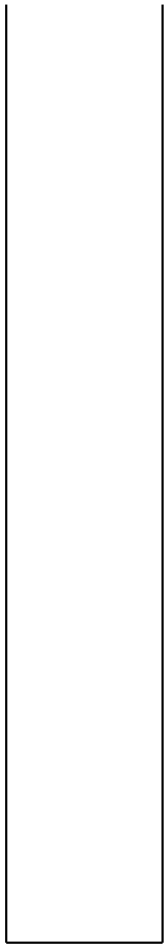
( ) (

Stacks
- Paranthesis Matching



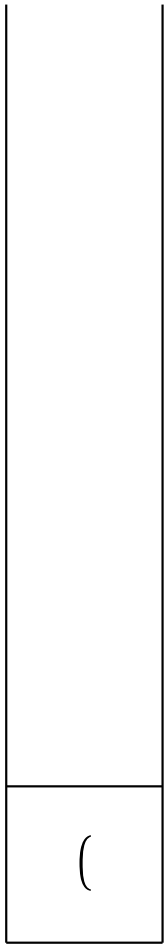
( ) (

Stacks
- Paranthesis Matching



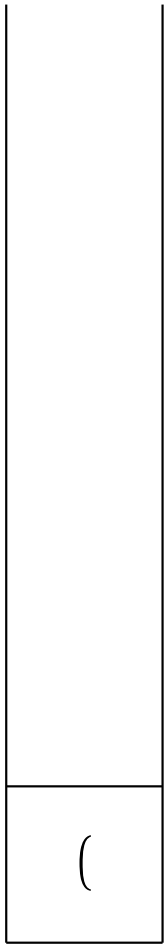
( ) (

Stacks
- Paranthesis Matching



( ) (

Stacks
- Paranthesis Matching



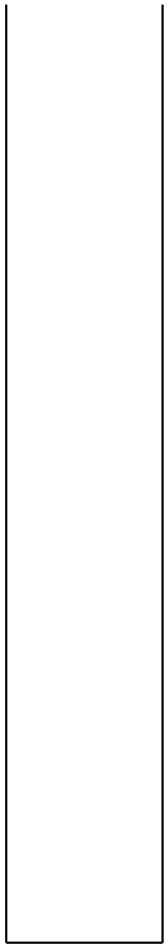
( ) (

Not Matching!

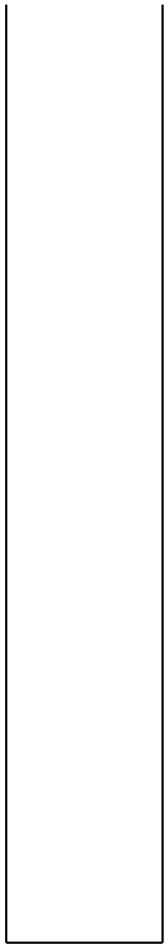
(i) Input read;
(ii) stack not empty.

Stacks
- Paranthesis Matching

Criteria



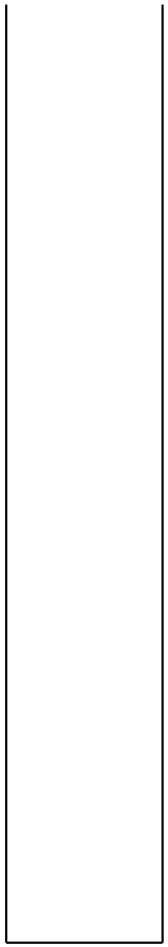
Stacks
- Paranthesis Matching



Criteria

- Input is completely read.

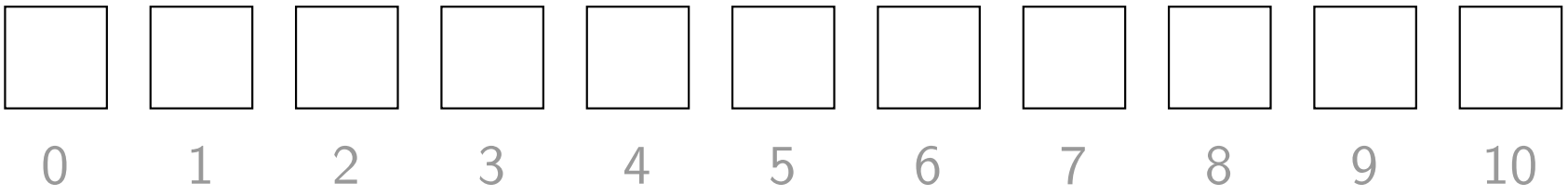
Stacks
- Paranthesis Matching



Criteria

- Input is completely read.
- Stack is empty.

Paranthesis Matching - Arrays



Paranthesis Matching - Arrays

(	(	)	)							
0	1	2	3	4	5	6	7	8	9	10

Paranthesis Matching - Arrays

(	(	)	)							
0	1	2	3	4	5	6	7	8	9	10

Counter = 0

Paranthesis Matching - Arrays

(	(	)	)							
0	1	2	3	4	5	6	7	8	9	10

Counter = 1

Paranthesis Matching - Arrays

(	(	)	)							
0	1	2	3	4	5	6	7	8	9	10

Counter = 2

Paranthesis Matching - Arrays

(	(	)	)							
0	1	2	3	4	5	6	7	8	9	10

Counter = 1

Paranthesis Matching - Arrays

(	(	)	)							
0	1	2	3	4	5	6	7	8	9	10

Counter = 0

Matching

Paranthesis Matching - Arrays

(	(	)	)	(	(	)	)			
0	1	2	3	4	5	6	7	8	9	10

Counter = 0

Paranthesis Matching - Arrays

(	(	)	)	(	(	)	)			
0	1	2	3	4	5	6	7	8	9	10

Counter = 1

Paranthesis Matching - Arrays

(	(	)	)	(	(	)	)			
0	1	2	3	4	5	6	7	8	9	10

Counter = 2

Paranthesis Matching - Arrays

(	(	)	)	(	(	)	)			
0	1	2	3	4	5	6	7	8	9	10

Counter = 1

Paranthesis Matching - Arrays

(	(	)	)	(	(	)	)			
0	1	2	3	4	5	6	7	8	9	10

Counter = 0

Paranthesis Matching - Arrays

(	(	)	)	(	(	)	)			
0	1	2	3	4	5	6	7	8	9	10

Counter = 1

Paranthesis Matching - Arrays

(	(	)	)	(	(	)	)			
0	1	2	3	4	5	6	7	8	9	10

Counter = 2

Paranthesis Matching - Arrays

(	(	)	)	(	(	)	)			
0	1	2	3	4	5	6	7	8	9	10

Counter = 1

Paranthesis Matching - Arrays

(	(	)	)	(	(	)	)			
0	1	2	3	4	5	6	7	8	9	10

Counter = 0

Matching

Paranthesis Matching - Arrays

(	(	)								
0	1	2	3	4	5	6	7	8	9	10

Counter = 0

Paranthesis Matching - Arrays

(	(	)								
0	1	2	3	4	5	6	7	8	9	10

Counter = 1

Paranthesis Matching - Arrays

(	(	)								
0	1	2	3	4	5	6	7	8	9	10

Counter = 2

Paranthesis Matching - Arrays

(	(	)								
0	1	2	3	4	5	6	7	8	9	10

Counter = 1

Not matching

Paranthesis Matching - Arrays

)	(									
0	1	2	3	4	5	6	7	8	9	10

Counter = 0

Paranthesis Matching - Arrays

)	(									
0	1	2	3	4	5	6	7	8	9	10

Counter $= -1$

Paranthesis Matching - Arrays

)	(									
0	1	2	3	4	5	6	7	8	9	10

Counter = 0

Not matching

Paranthesis Matching - Arrays

)	(									
0	1	2	3	4	5	6	7	8	9	10

Counter ≥ 0

Stacks
- Application 2

Stacks
- Application 2

Multiple Paranthesis Matching

Same as the previous problem, except we have ( ), { }, and [ ].

Stacks
- Application 2

Multiple Paranthesis Matching

Same as the previous problem, except we have ( ), { }, and [ ].

Examples

Stacks
- Application 2

Multiple Paranthesis Matching

Same as the previous problem, except we have ( ), { }, and [ ].

Examples

[{ ( ) }]

Stacks
- Application 2

Multiple Paranthesis Matching

Same as the previous problem, except we have ( ), { }, and [ ].

Examples

[{ ( ) }]

[{ } ({ })]

Multiple Paranthesis Matching

Same as the previous problem, except we have ( ), { }, and [ ].

Examples

[{ ( ) }]

[{ } ({ })]

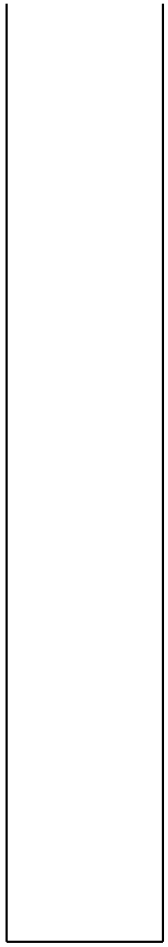
[{] }

Stacks

- Multiple Paranthesis Matching

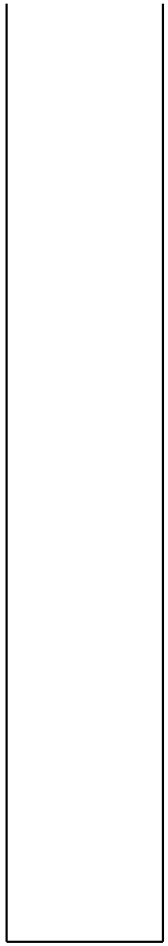
Stacks

- Multiple Paranthesis Matching



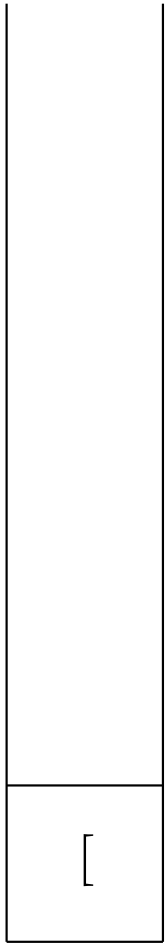
Stacks

- Multiple Paranthesis Matching



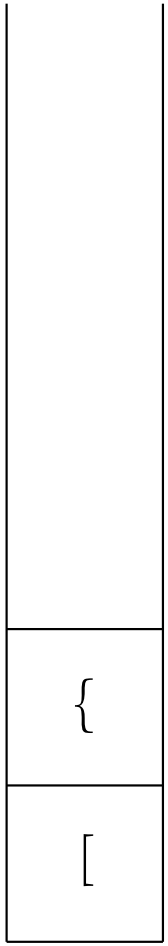
[{ ( ) }]

Stacks
- Multiple Paranthesis Matching



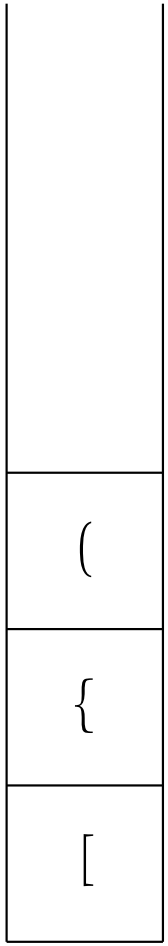
[{ ( ) }]

Stacks
- Multiple Paranthesis Matching



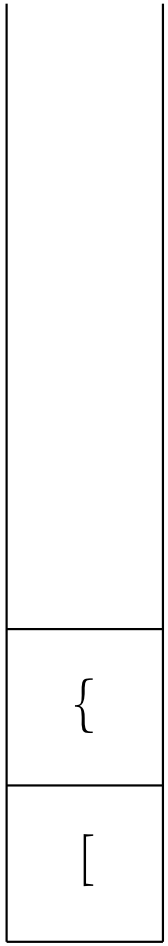
[{ ( ) }]

Stacks
- Multiple Paranthesis Matching



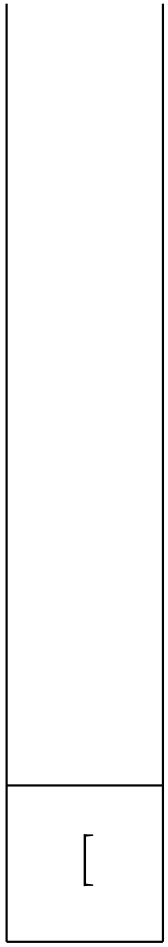
[{ ( ) }]

Stacks
- Multiple Paranthesis Matching



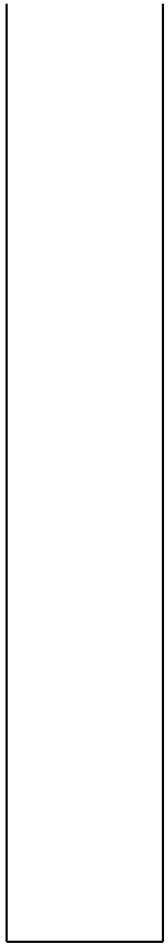
[{ ( ) }]

Stacks
- Multiple Paranthesis Matching



[{ ( ) }]

Stacks
- Multiple Paranthesis Matching

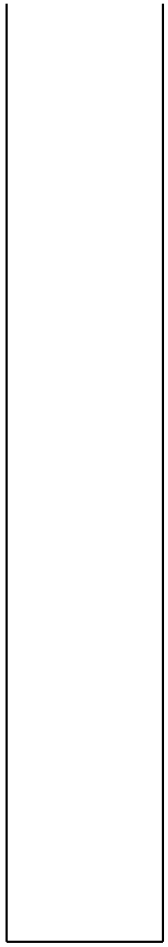


[{ ( ) }]

Matching

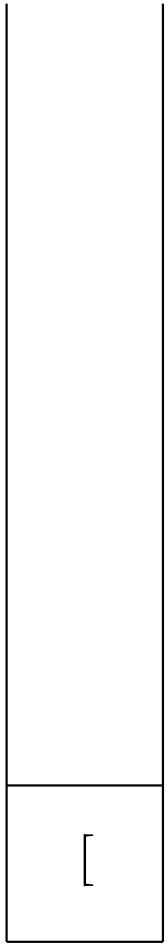
Stacks

- Multiple Paranthesis Matching



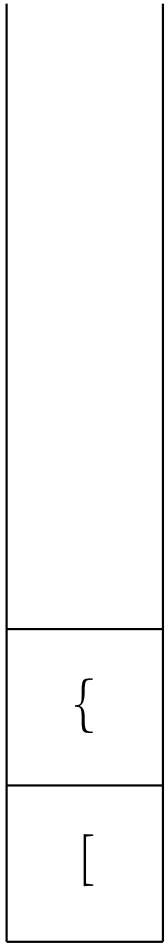
[{ } ({ })]

Stacks
- Multiple Paranthesis Matching



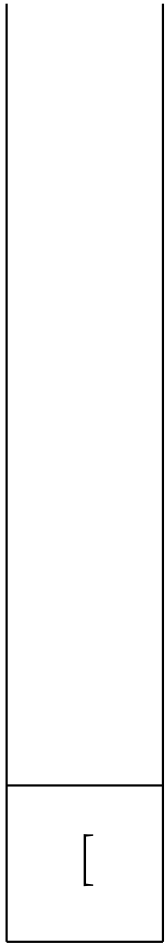
[{ } ({ })]

Stacks
- Multiple Paranthesis Matching



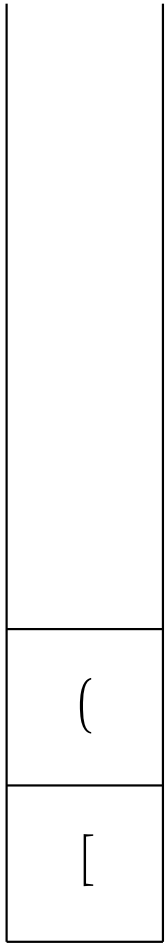
[{ } ({ })]

Stacks
- Multiple Paranthesis Matching



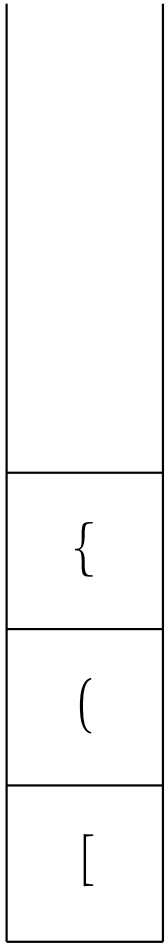
[{ } ({ })]

Stacks
- Multiple Paranthesis Matching



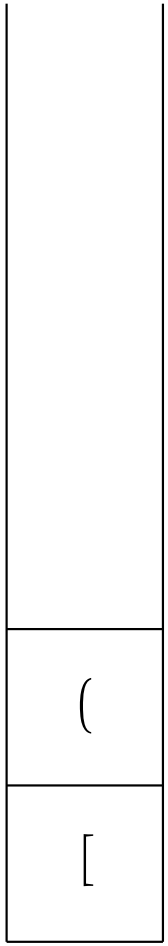
[{ } ({ })]

Stacks
- Multiple Paranthesis Matching



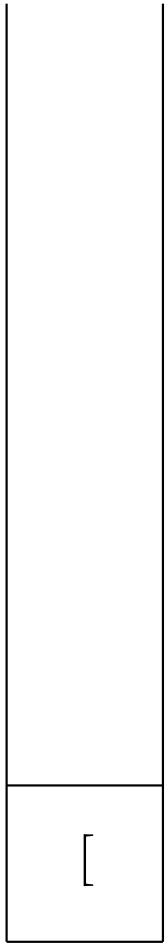
[{ } ({ })]

Stacks
- Multiple Paranthesis Matching



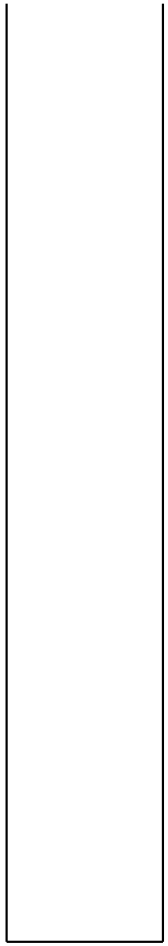
[{ } ({ })]

Stacks
- Multiple Paranthesis Matching



[{ } ({ })]

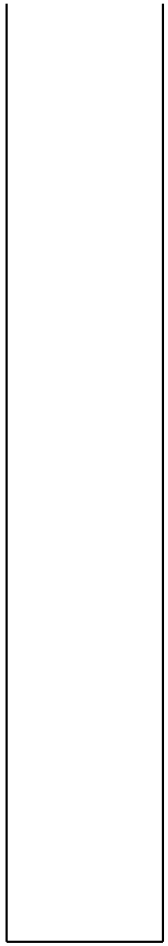
Stacks
- Multiple Paranthesis Matching



[{ } ({ })]

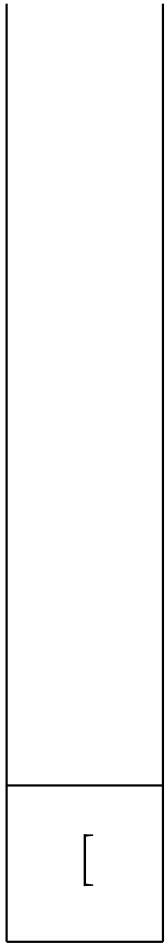
Matching

Stacks
- Multiple Paranthesis Matching



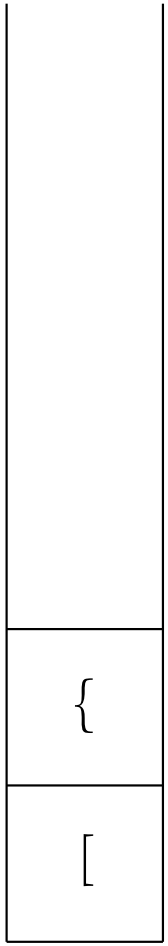
[{] }

Stacks
- Multiple Paranthesis Matching



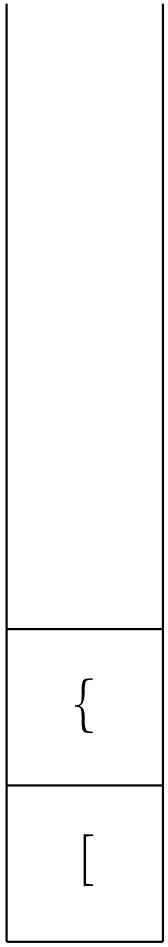
[{] }

Stacks
- Multiple Paranthesis Matching



[{] }

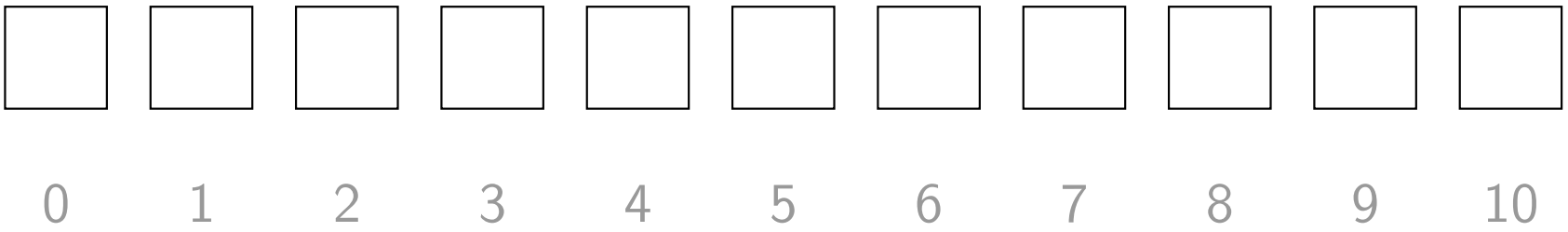
Stacks
- Multiple Paranthesis Matching



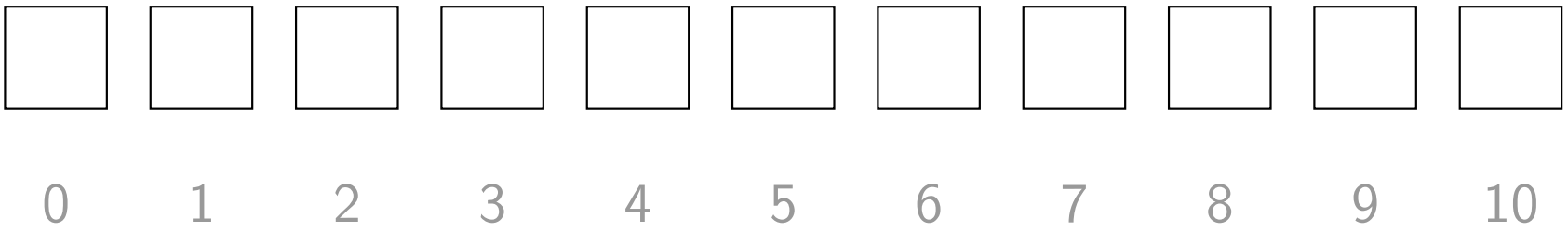
[{] }

Not Matching

Multiple Paranthesis Matching - Arrays



Multiple Paranthesis Matching - Arrays

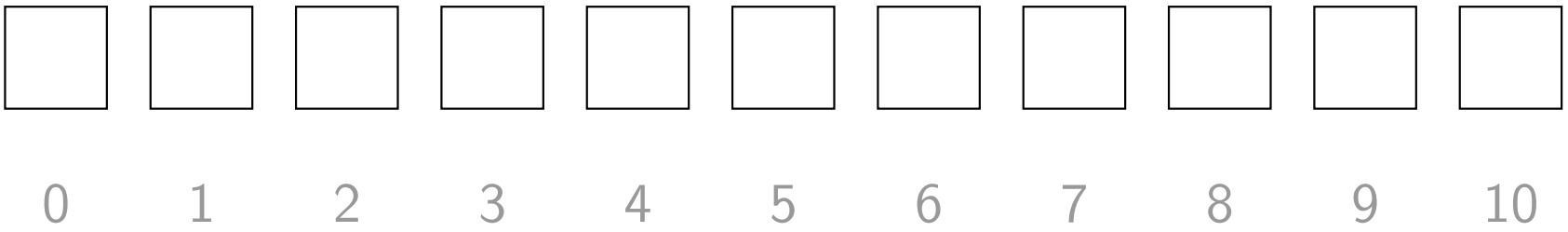


[Counter1 ≥ 0

{ Counter2 ≥ 0

(Counter3 ≥ 0

Multiple Paranthesis Matching - Arrays



[Counter1 ≥ 0

{ Counter2 ≥ 0

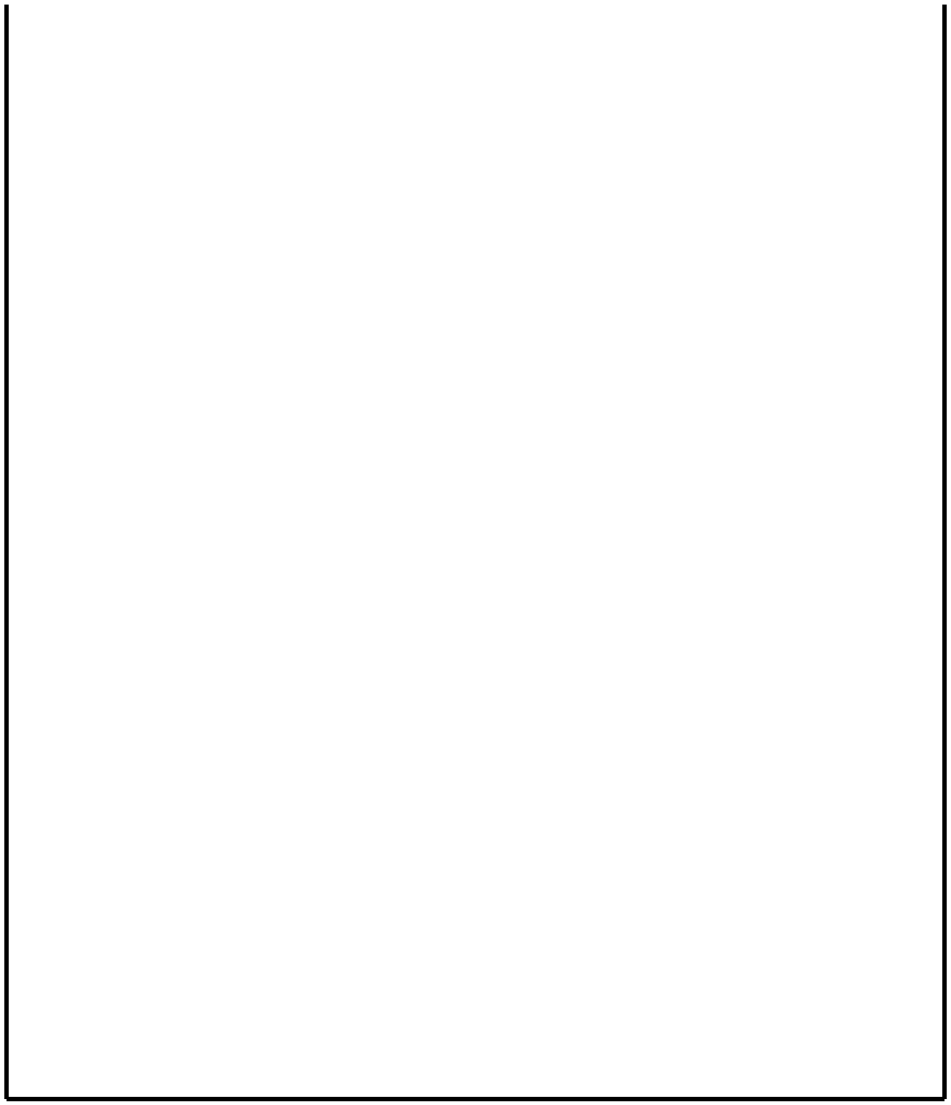
(Counter3 ≥ 0

How to handle [{] } ?

Stacks – Recursion

```
recurse(int n) {
  if (n > 1) recurse (n-1);
  printf("%d", n);
}
```

Stacks – Recursion



```
recurse(int n) {
  if (n > 1) recurse (n-1);
  printf("%d", n);
}
```

Stacks – Recursion

```
recurse(int 3) {
  if (3 > 1) recurse (3-1);
  printf("%d", 3);
}
```

Stacks – Recursion

```
recurse(int 3) {
  if (3 > 1) recurse (3-1);
  printf("%d", 3);
}
```

Stacks – Recursion

```
recurse(int 3) {
  if (3 > 1) recurse (3-1);
  printf("%d", 3);
}
```

Stacks – Recursion

```
recurse(int 2) {
  if (2 > 1) recurse (2-1);
  printf("%d", 2);
}
```

```
recurse(int 3) {
  if (3 > 1) recurse (3-1);
  printf("%d", 3);
}
```

Stacks – Recursion

```
recurse(int 2) {
  if (2 > 1) recurse (2-1);
  printf("%d", 2);
}
```

```
recurse(int 3) {
  if (3 > 1) recurse (3-1);
  printf("%d", 3);
}
```

Stacks – Recursion

```
recurse(int 2) {
  if (2 > 1) recurse (2-1);
  printf("%d", 2);
}
```

```
recurse(int 3) {
  if (3 > 1) recurse (3-1);
  printf("%d", 3);
}
```

Stacks – Recursion

```
recurse(int 2) {
  if (2 > 1) recurse (2-1);
  printf("%d", 2);
}
```

```
recurse(int 3) {
  if (3 > 1) recurse (3-1);
  printf("%d", 3);
}
```

```
recurse(int 1) {
  if (1 > 1) recurse (1-1);
  printf("%d", 1);
}
```

Stacks – Recursion

```
recurse(int 2) {
  if (2 > 1) recurse (2-1);
  printf("%d", 2);
}
```

```
recurse(int 3) {
  if (3 > 1) recurse (3-1);
  printf("%d", 3);
}
```

```
recurse(int 1) {
  if (1 > 1) recurse (1-1);
  printf("%d", 1);
}
```

Stacks – Recursion

```
recurse(int 2) {
  if (2 > 1) recurse (2-1);
  printf("%d", 2);
}
```

```
recurse(int 3) {
  if (3 > 1) recurse (3-1);
  printf("%d", 3);
}
```

```
recurse(int 1) {
  if (1 > 1) recurse (1-1);
  printf("%d", 1);
}
```

1

```
recurse(int 2) {
  if (2 > 1) recurse (2-1);
  printf("%d", 2);
}
```

```
recurse(int 3) {
  if (3 > 1) recurse (3-1);
  printf("%d", 3);
}
```

Stacks – Recursion

```
recurse(int 2) {
  if (2 > 1) recurse (2-1);
  printf("%d", 2);
}
```

```
recurse(int 3) {
  if (3 > 1) recurse (3-1);
  printf("%d", 3);
}
```

1

Stacks – Recursion

```
recurse(int 2) {
  if (2 > 1) recurse (2-1);
  printf("%d", 2);
}
```

```
recurse(int 3) {
  if (3 > 1) recurse (3-1);
  printf("%d", 3);
}
```

1

Stacks – Recursion

```
recurse(int 2) {
  if (2 > 1) recurse (2-1);
  printf("%d", 2);
}
```

```
recurse(int 3) {
  if (3 > 1) recurse (3-1);
  printf("%d", 3);
}
```

1 2

Stacks – Recursion

```
recurse(int 3) {
  if (3 > 1) recurse (3-1);
  printf("%d", 3);
}
```

1 2

Stacks – Recursion

```
recurse(int 3) {
  if (3 > 1) recurse (3-1);
  printf("%d", 3);
}
```

1 2

Stacks – Recursion

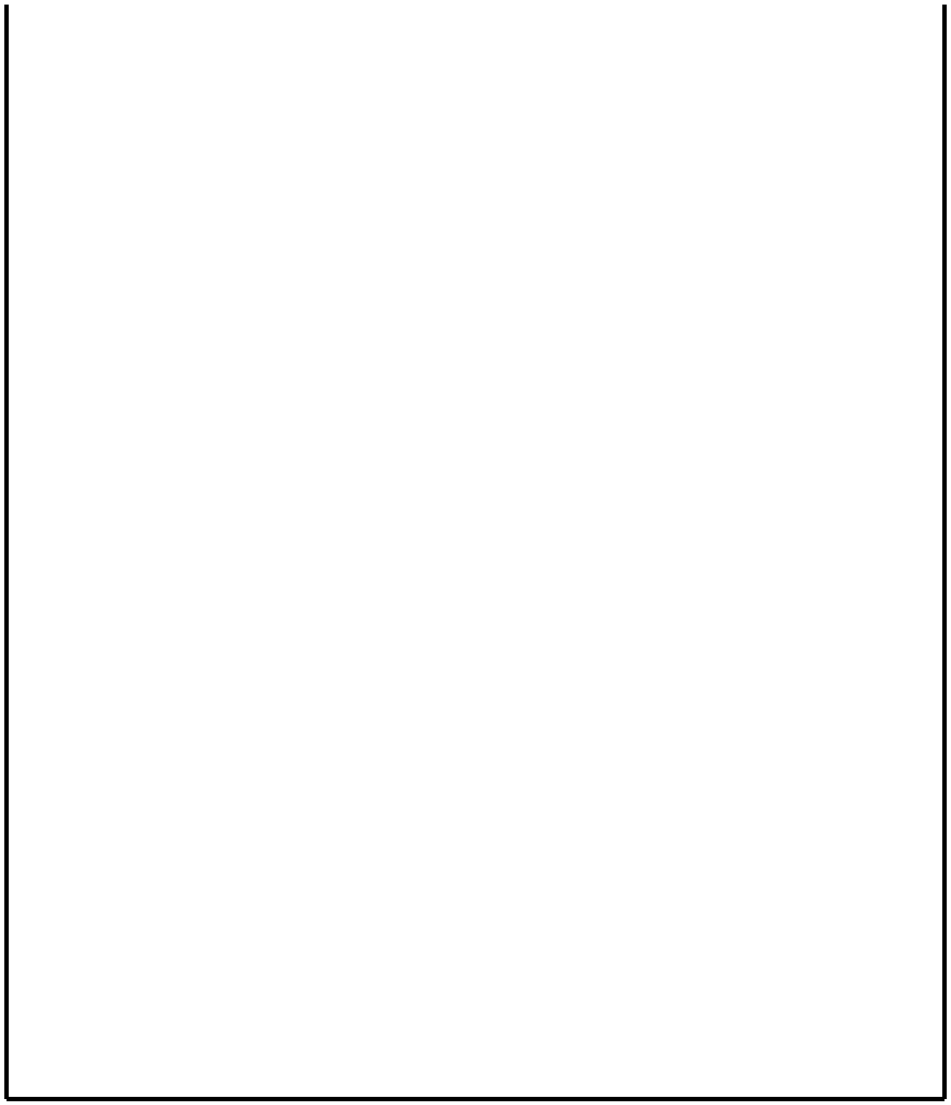
```
recurse(int 3) {
  if (3 > 1) recurse (3-1);
  printf("%d", 3);
}
```

1 2

Stacks – Recursion

```
recurse(int 3) {
  if (3 > 1) recurse (3-1);
  printf("%d", 3);
}
```

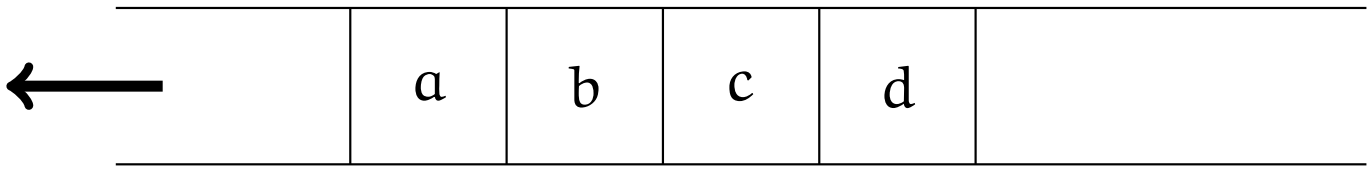
```
1 2 3
```



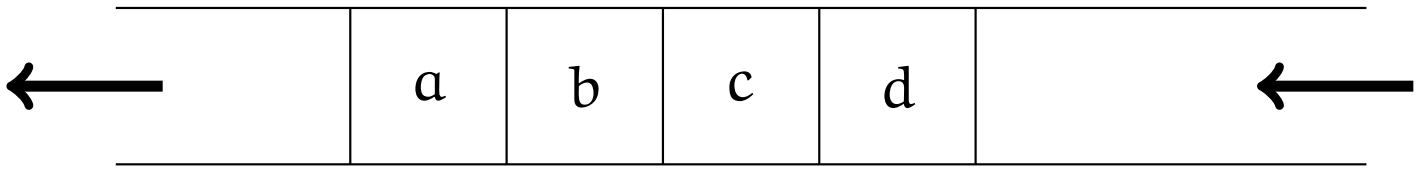
1 2 3

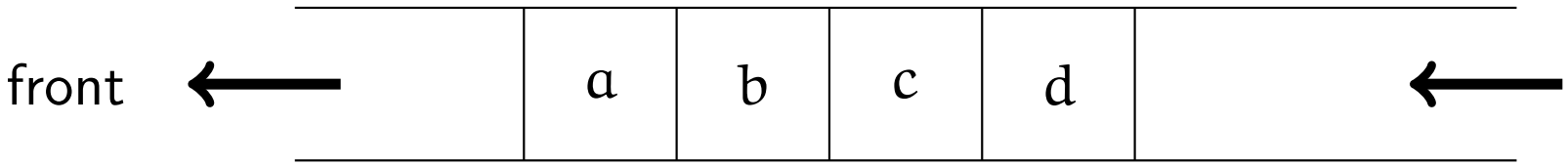
Data Structures
- Queues

	a	b	c	d	

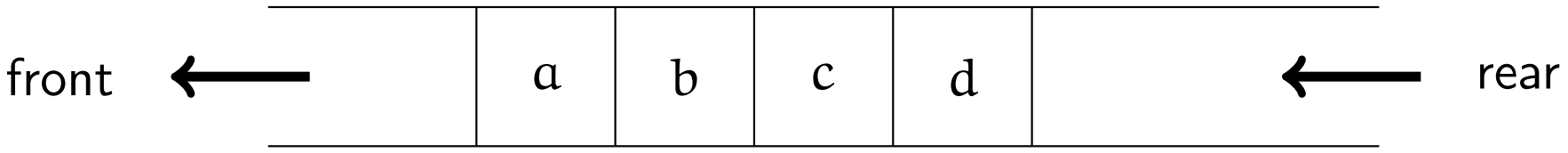

a
b
c
d

Data Structures - Queues

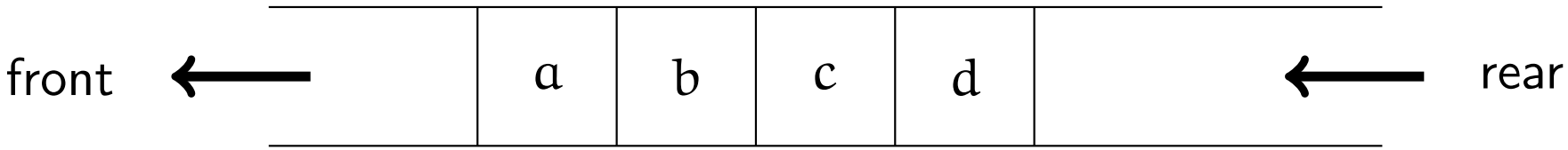



front
a
b
c
d

Data Structures - Queues

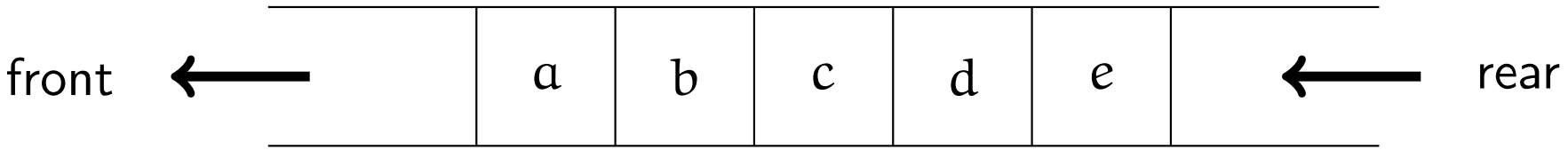


Data Structures - Queues



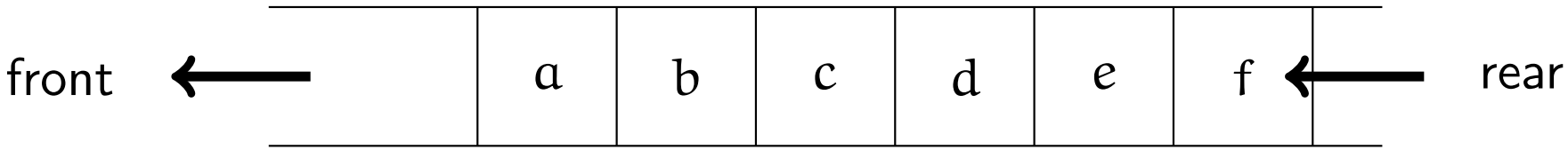
Store data

Data Structures - Queues



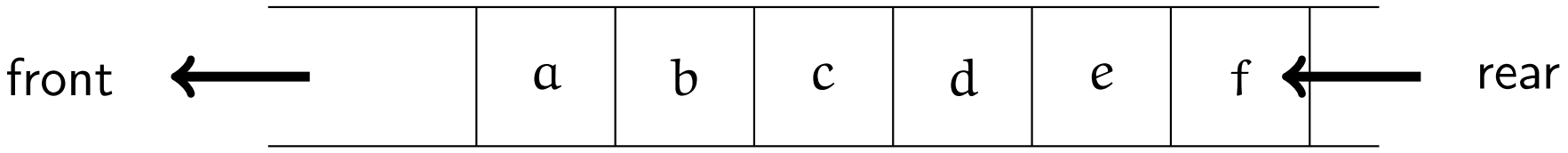
Store data

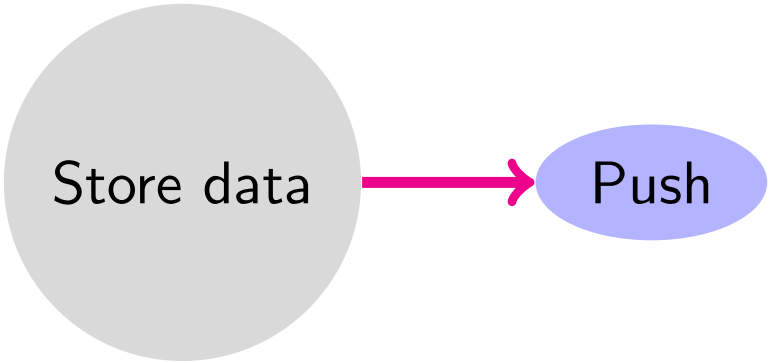
Data Structures - Queues

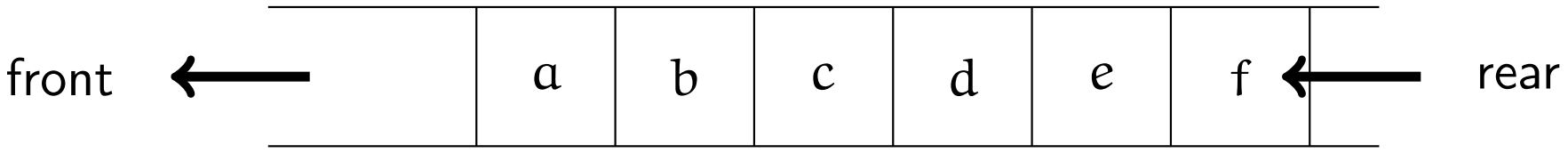


Store data

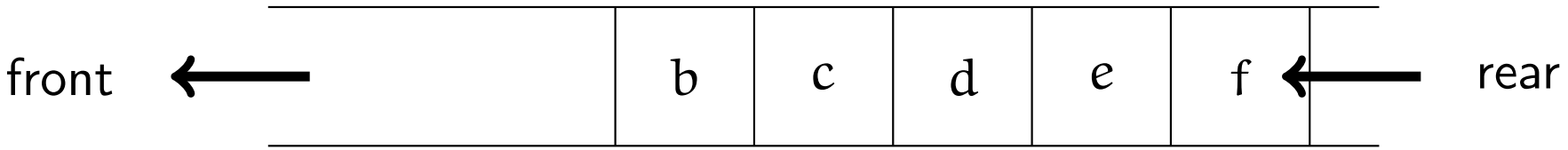
Data Structures - Queues



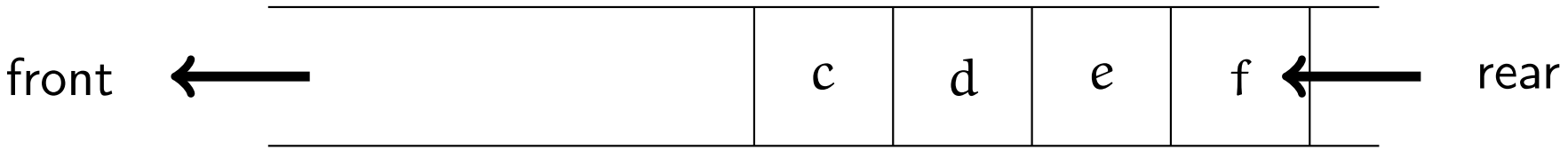




Retrieve Data



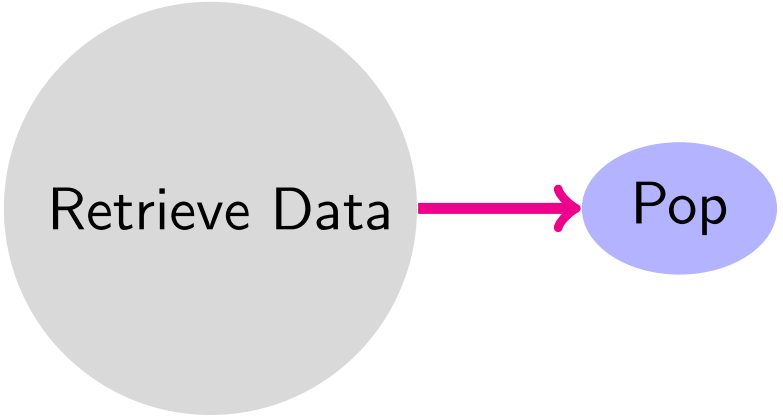
Retrieve Data



Retrieve Data

Retrieve Data


front
d
e
f
rear


Retrieve Data
Pop

Queues
- Applications

Queues
- Applications

- Unfortunately, no easy to implement application.

Queues
- Applications

- Unfortunately, no easy to implement application.
- Operating systems : job queue

Queues
- Applications

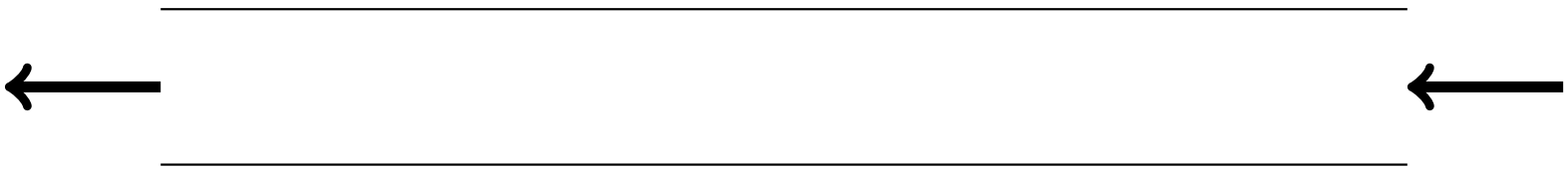
- Unfortunately, no easy to implement application.
- Operating systems : job queue
- Printer spool

Queues
- Applications

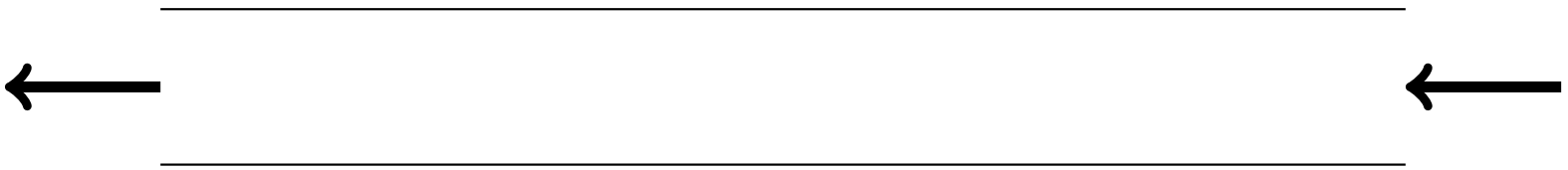
- Unfortunately, no easy to implement application.
- Operating systems : job queue
- Printer spool
- Keyboard buffer

Queues
- Implementation

Queues
- Implementation



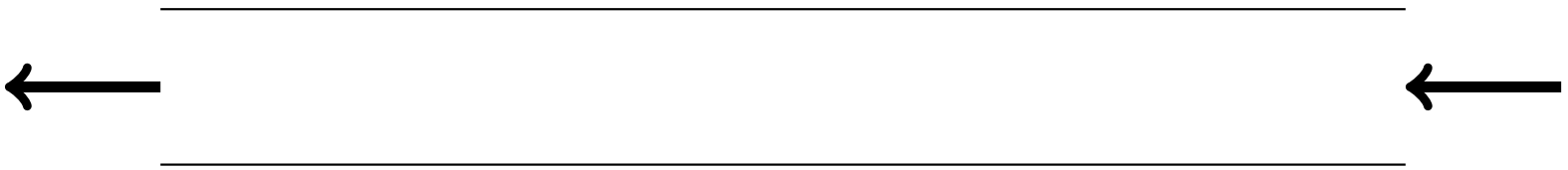
Queues
- Implementation



`Input :`

`Output :`

Queues
- Implementation



Input : abcPPdfaPPPPP

Output :

Queues
- Implementation

Input : abcPPdfaPPPPP

Output :

Queues
- Implementation

Input : abcPPdfaPPPPP

Output :

Queues
- Implementation

Input : abcPPdfaPPPPP

Output :

Queues
- Implementation

Input : abcPPdfaPPPPP

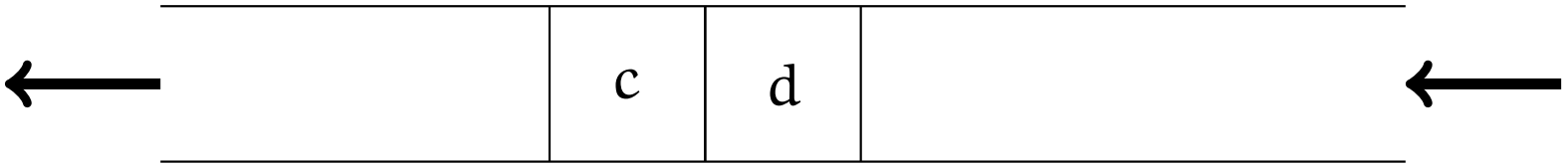
Output : a

Queues
- Implementation

Input : abcPPdfaPPPPP

Output : ab

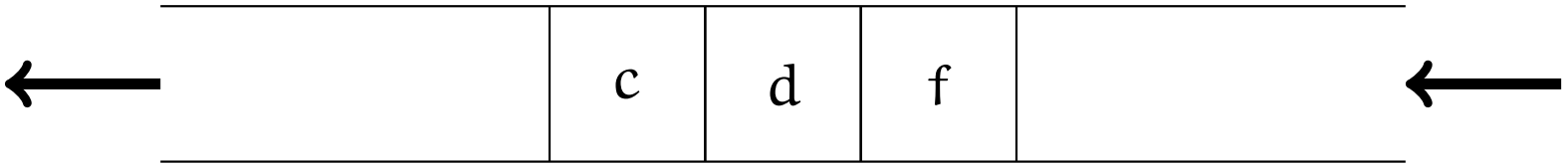
Queues
- Implementation



Input : abcPPdfaPPPPP

Output : ab

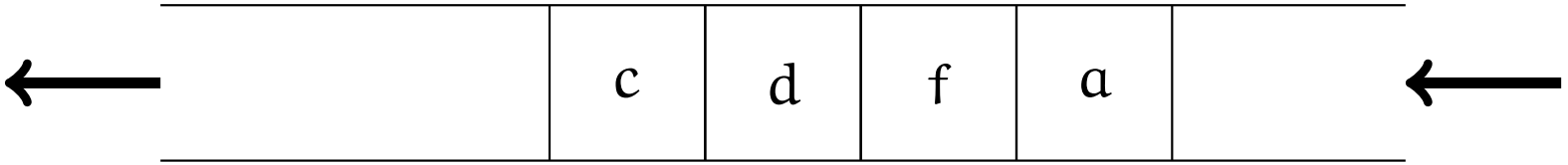
Queues
- Implementation



Input : abcPPdfaPPPPP

Output : ab

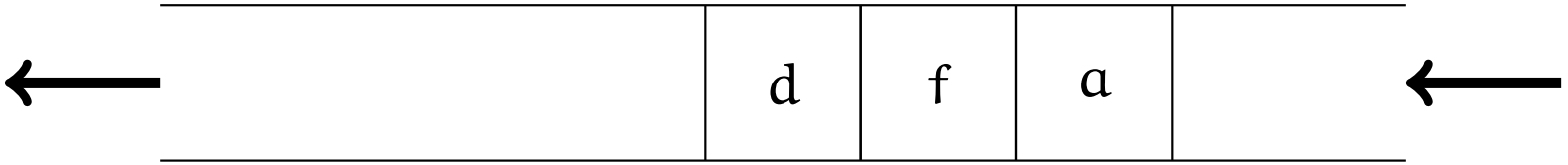
Queues
- Implementation



Input : abcPPdfaPPPPP

Output : ab

Queues
- Implementation



Input : abcPPdfaPPPPP

Output : abc

Queues
- Implementation

Input : abcPPdfaPPPPP

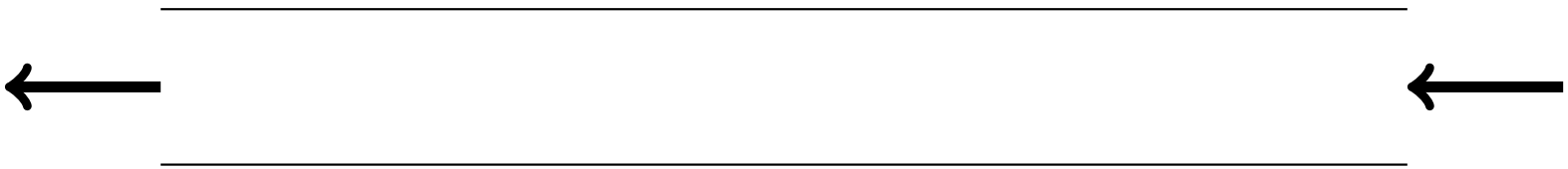
Output : abcd

Queues
- Implementation

Input : abcPPdfaPPPPP

Output : abcdf

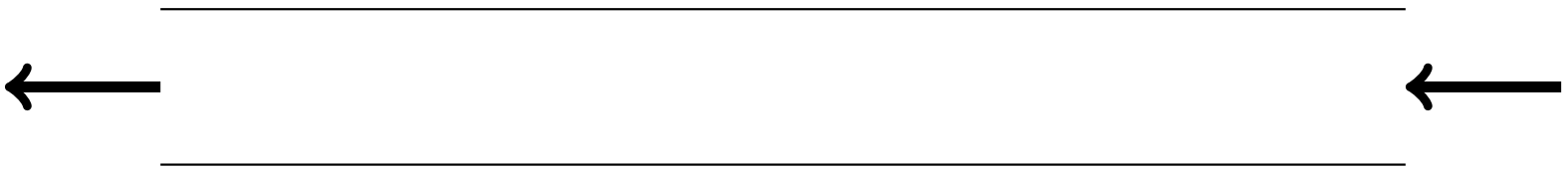
Queues
- Implementation



Input : abcPPdfaPPPPP

Output : abcdfa

Queues
- Implementation



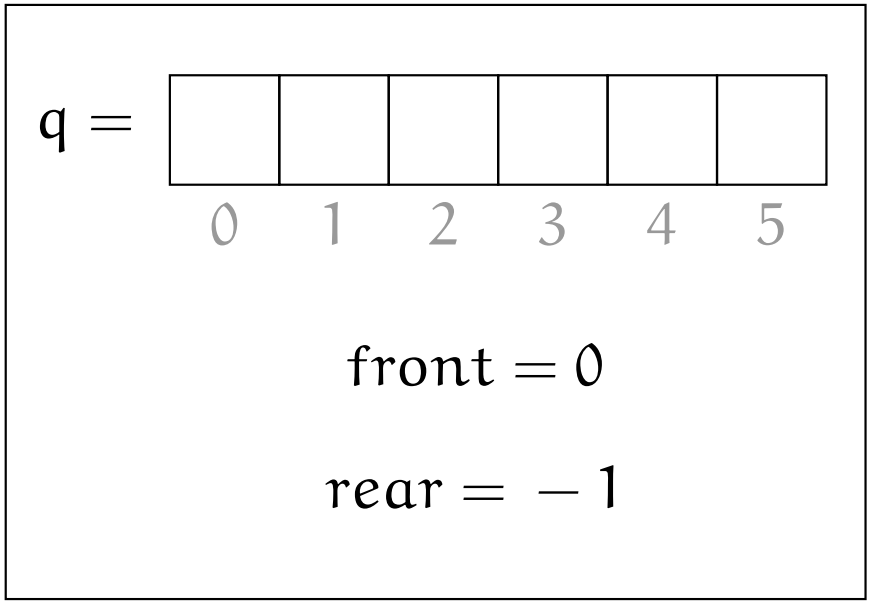
Input : abcPPdfaPPPPP

Output : abcdfaE

Queues
- Implementation

Queues
- Implementation

Queue



Queues
- Implementation

Queue

$q =$

0	1	2	3	4	5

$\mathrm{front} = 0$

$\mathrm{rear} = -1$

Push

$\mathrm{rear}++;$
$q[\mathrm{rear}] = a;$

Queues
- Implementation

Queue

$q =$

0	1	2	3	4	5

$\mathrm{front} = 0$

$\mathrm{rear} = -1$

Push

$\mathrm{rear}++;$
$q[\mathrm{rear}] = a;$

Pop

$t = q[\mathrm{front}];$
$\mathrm{front}++;$