Problem 1670: Design Front Middle Back Queue

Problem Information

Difficulty: Medium
Acceptance Rate: 0.00%
Paid Only: No

Problem Description

Design a queue that supports

push

and

pop

operations in the front, middle, and back.

Implement the

FrontMiddleBack

class:

FrontMiddleBack()

Initializes the queue.

void pushFront(int val)

Adds

val

to the

front

of the queue.

void pushMiddle(int val)

Adds

val

to the

middle

of the queue.

void pushBack(int val)

Adds

val

to the

back

of the queue.

int popFront()

Removes the

front

element of the queue and returns it. If the queue is empty, return

-1

.

int popMiddle()

Removes the

middle

element of the queue and returns it. If the queue is empty, return

-1

.

int popBack()

Removes the

back

element of the queue and returns it. If the queue is empty, return

-1

.

Notice

that when there are

two

middle position choices, the operation is performed on the

frontmost

middle position choice. For example:

Pushing

6

into the middle of

[1, 2, 3, 4, 5]

results in

[1, 2,

6

, 3, 4, 5]

.

Popping the middle from

[1, 2,

3

, 4, 5, 6]

returns

3

and results in

[1, 2, 4, 5, 6]

.

Example 1:

Input:

["FrontMiddleBackQueue", "pushFront", "pushBack", "pushMiddle", "pushMiddle", "popFront", "popMiddle", "popMiddle", "popBack", "popFront"] [[], [1], [2], [3], [4], [], [], [], [], []]

Output:

[null, null, null, null, null, 1, 3, 4, 2, -1]

Explanation:

FrontMiddleBackQueue q = new FrontMiddleBackQueue(); q.pushFront(1); // [

1

] q.pushBack(2); // [1,

2

] q.pushMiddle(3); // [1,

3

, 2] q.pushMiddle(4); // [1,

4

, 3, 2] q.popFront(); // return 1 -> [4, 3, 2] q.popMiddle(); // return 3 -> [4, 2] q.popMiddle(); // return 4 -> [2] q.popBack(); // return 2 -> [] q.popFront(); // return -1 -> [] (The queue is empty)

Constraints:

1 <= val <= 10

9

At most

1000

calls will be made to

pushFront

,

pushMiddle

,

pushBack

,

popFront

,

popMiddle

, and

popBack

.

Code Snippets

C++:

```
class FrontMiddleBackQueue {
public:
FrontMiddleBackQueue() {

}

void pushFront(int val) {

}
```

```
void pushMiddle(int val) {

}

void pushBack(int val) {

}

int popFront() {

}

int popMiddle() {

}

int popBack() {

}
};

/**
* Your FrontMiddleBackQueue object will be instantiated and called as such:
* FrontMiddleBackQueue* obj = new FrontMiddleBackQueue();
* obj->pushFront(val);
* obj->pushMiddle(val);
* obj->pushBack(val);
* int param_4 = obj->popFront();
* int param_5 = obj->popMiddle();
* int param_6 = obj->popBack();
*/
```

Java:

```
class FrontMiddleBackQueue {

public FrontMiddleBackQueue() {

}

public void pushFront(int val) {
```

```
}

public void pushMiddle(int val) {

}

public void pushBack(int val) {

}

public int popFront() {

}

public int popMiddle() {

}

public int popBack() {

}
}

/**
* Your FrontMiddleBackQueue object will be instantiated and called as such:
* FrontMiddleBackQueue obj = new FrontMiddleBackQueue();
* obj.pushFront(val);
* obj.pushMiddle(val);
* obj.pushBack(val);
* int param_4 = obj.popFront();
* int param_5 = obj.popMiddle();
* int param_6 = obj.popBack();
*/
```

Python3:

```
class FrontMiddleBackQueue:

def __init__(self):


def pushFront(self, val: int) -> None:
```

```
def pushMiddle(self, val: int) -> None:


def pushBack(self, val: int) -> None:


def popFront(self) -> int:


def popMiddle(self) -> int:


def popBack(self) -> int:



# Your FrontMiddleBackQueue object will be instantiated and called as such:
# obj = FrontMiddleBackQueue()
# obj.pushFront(val)
# obj.pushMiddle(val)
# obj.pushBack(val)
# param_4 = obj.popFront()
# param_5 = obj.popMiddle()
# param_6 = obj.popBack()
```

Python:

```
class FrontMiddleBackQueue(object):

def __init__(self):


def pushFront(self, val):
"""
:type val: int
:rtype: None
"""


def pushMiddle(self, val):
```

```
        """
        :type val: int
        :rtype: None
        """


    def pushBack(self, val):
        """
        :type val: int
        :rtype: None
        """


    def popFront(self):
        """
        :rtype: int
        """


    def popMiddle(self):
        """
        :rtype: int
        """


    def popBack(self):
        """
        :rtype: int
        """



# Your FrontMiddleBackQueue object will be instantiated and called as such:
# obj = FrontMiddleBackQueue()
# obj.pushFront(val)
# obj.pushMiddle(val)
# obj.pushBack(val)
# param_4 = obj.popFront()
# param_5 = obj.popMiddle()
# param_6 = obj.popBack()
```

JavaScript:

```
var FrontMiddleBackQueue = function() {

};

/**
* @param {number} val
* @return {void}
*/
FrontMiddleBackQueue.prototype.pushFront = function(val) {

};

/**
* @param {number} val
* @return {void}
*/
FrontMiddleBackQueue.prototype.pushMiddle = function(val) {

};

/**
* @param {number} val
* @return {void}
*/
FrontMiddleBackQueue.prototype.pushBack = function(val) {

};

/**
* @return {number}
*/
FrontMiddleBackQueue.prototype.popFront = function() {

};

/**
* @return {number}
*/
FrontMiddleBackQueue.prototype.popMiddle = function() {
```

```
};

/**
* @return {number}
*/
FrontMiddleBackQueue.prototype.popBack = function() {

};

/**
* Your FrontMiddleBackQueue object will be instantiated and called as such:
* var obj = new FrontMiddleBackQueue()
* obj.pushFront(val)
* obj.pushMiddle(val)
* obj.pushBack(val)
* var param_4 = obj.popFront()
* var param_5 = obj.popMiddle()
* var param_6 = obj.popBack()
*/
```

TypeScript:

```
class FrontMiddleBackQueue {
constructor() {

}

pushFront(val: number): void {

}

pushMiddle(val: number): void {

}

pushBack(val: number): void {

}

popFront(): number {

}
```

```
popMiddle(): number {

}

popBack(): number {

}
}

/**
* Your FrontMiddleBackQueue object will be instantiated and called as such:
* var obj = new FrontMiddleBackQueue()
* obj.pushFront(val)
* obj.pushMiddle(val)
* obj.pushBack(val)
* var param_4 = obj.popFront()
* var param_5 = obj.popMiddle()
* var param_6 = obj.popBack()
*/
```

C#:

```
public class FrontMiddleBackQueue {

public FrontMiddleBackQueue() {

}

public void PushFront(int val) {

}

public void PushMiddle(int val) {

}

public void PushBack(int val) {

}

public int PopFront() {
```

```
}

public int PopMiddle() {

}

public int PopBack() {

}
}

/**
* Your FrontMiddleBackQueue object will be instantiated and called as such:
* FrontMiddleBackQueue obj = new FrontMiddleBackQueue();
* obj.PushFront(val);
* obj.PushMiddle(val);
* obj.PushBack(val);
* int param_4 = obj.PopFront();
* int param_5 = obj.PopMiddle();
* int param_6 = obj.PopBack();
*/
```

C:

```
typedef struct {

} FrontMiddleBackQueue;


FrontMiddleBackQueue* frontMiddleBackQueueCreate() {

}

void frontMiddleBackQueuePushFront(FrontMiddleBackQueue* obj, int val) {

}

void frontMiddleBackQueuePushMiddle(FrontMiddleBackQueue* obj, int val) {
```

```
}

void frontMiddleBackQueuePushBack(FrontMiddleBackQueue* obj, int val) {

}

int frontMiddleBackQueuePopFront(FrontMiddleBackQueue* obj) {

}

int frontMiddleBackQueuePopMiddle(FrontMiddleBackQueue* obj) {

}

int frontMiddleBackQueuePopBack(FrontMiddleBackQueue* obj) {

}

void frontMiddleBackQueueFree(FrontMiddleBackQueue* obj) {

}

/**
* Your FrontMiddleBackQueue struct will be instantiated and called as such:
* FrontMiddleBackQueue* obj = frontMiddleBackQueueCreate();
* frontMiddleBackQueuePushFront(obj, val);

* frontMiddleBackQueuePushMiddle(obj, val);

* frontMiddleBackQueuePushBack(obj, val);

* int param_4 = frontMiddleBackQueuePopFront(obj);

* int param_5 = frontMiddleBackQueuePopMiddle(obj);

* int param_6 = frontMiddleBackQueuePopBack(obj);

* frontMiddleBackQueueFree(obj);
*/
```

Go:

```
type FrontMiddleBackQueue struct {

}


func Constructor() FrontMiddleBackQueue {

}


func (this *FrontMiddleBackQueue) PushFront(val int) {

}


func (this *FrontMiddleBackQueue) PushMiddle(val int) {

}


func (this *FrontMiddleBackQueue) PushBack(val int) {

}


func (this *FrontMiddleBackQueue) PopFront() int {

}


func (this *FrontMiddleBackQueue) PopMiddle() int {

}


func (this *FrontMiddleBackQueue) PopBack() int {

}


/**
```

```
* Your FrontMiddleBackQueue object will be instantiated and called as such:
* obj := Constructor();
* obj.PushFront(val);
* obj.PushMiddle(val);
* obj.PushBack(val);
* param_4 := obj.PopFront();
* param_5 := obj.PopMiddle();
* param_6 := obj.PopBack();
*/
```

Kotlin:

```
class FrontMiddleBackQueue() {

fun pushFront(`val`: Int) {

}

fun pushMiddle(`val`: Int) {

}

fun pushBack(`val`: Int) {

}

fun popFront(): Int {

}

fun popMiddle(): Int {

}

fun popBack(): Int {

}

}

/**
* Your FrontMiddleBackQueue object will be instantiated and called as such:
```

```
* var obj = FrontMiddleBackQueue()
* obj.pushFront(`val`)
* obj.pushMiddle(`val`)
* obj.pushBack(`val`)
* var param_4 = obj.popFront()
* var param_5 = obj.popMiddle()
* var param_6 = obj.popBack()
*/
```

Swift:

```
class FrontMiddleBackQueue {

init() {

}

func pushFront(_ val: Int) {

}

func pushMiddle(_ val: Int) {

}

func pushBack(_ val: Int) {

}

func popFront() -> Int {

}

func popMiddle() -> Int {

}

func popBack() -> Int {

}
}
```

```
/**
* Your FrontMiddleBackQueue object will be instantiated and called as such:
* let obj = FrontMiddleBackQueue()
* obj.pushFront(val)
* obj.pushMiddle(val)
* obj.pushBack(val)
* let ret_4: Int = obj.popFront()
* let ret_5: Int = obj.popMiddle()
* let ret_6: Int = obj.popBack()
*/
```

Rust:

```
struct FrontMiddleBackQueue {

}


/**
* `&self` means the method takes an immutable reference.
* If you need a mutable reference, change it to `&mut self` instead.
*/
impl FrontMiddleBackQueue {

fn new() -> Self {

}

fn push_front(&self, val: i32) {

}

fn push_middle(&self, val: i32) {

}

fn push_back(&self, val: i32) {

}

fn pop_front(&self) -> i32 {
```

```
}

fn pop_middle(&self) -> i32 {

}

fn pop_back(&self) -> i32 {

}
}

/**
* Your FrontMiddleBackQueue object will be instantiated and called as such:
* let obj = FrontMiddleBackQueue::new();
* obj.push_front(val);
* obj.push_middle(val);
* obj.push_back(val);
* let ret_4: i32 = obj.pop_front();
* let ret_5: i32 = obj.pop_middle();
* let ret_6: i32 = obj.pop_back();
*/
```

Ruby:

```
class FrontMiddleBackQueue
def initialize()

end


=begin
:type val: Integer
:rtype: Void
=end
def push_front(val)

end


=begin
:type val: Integer
```

```
:rtype: Void
=end
def push_middle(val)

end


=begin
:type val: Integer
:rtype: Void
=end
def push_back(val)

end


=begin
:rtype: Integer
=end
def pop_front()

end


=begin
:rtype: Integer
=end
def pop_middle()

end


=begin
:rtype: Integer
=end
def pop_back()

end


end
```

```
# Your FrontMiddleBackQueue object will be instantiated and called as such:
# obj = FrontMiddleBackQueue.new()
# obj.push_front(val)
# obj.push_middle(val)
# obj.push_back(val)
# param_4 = obj.pop_front()
# param_5 = obj.pop_middle()
# param_6 = obj.pop_back()
```

PHP:

```
class FrontMiddleBackQueue {
/**
*/
function __construct() {

}

/**
* @param Integer $val
* @return NULL
*/
function pushFront($val) {

}

/**
* @param Integer $val
* @return NULL
*/
function pushMiddle($val) {

}

/**
* @param Integer $val
* @return NULL
*/
function pushBack($val) {

}
```

```
/**
* @return Integer
*/
function popFront() {

}

/**
* @return Integer
*/
function popMiddle() {

}

/**
* @return Integer
*/
function popBack() {

}
}

/**
* Your FrontMiddleBackQueue object will be instantiated and called as such:
* $obj = FrontMiddleBackQueue();
* $obj->pushFront($val);
* $obj->pushMiddle($val);
* $obj->pushBack($val);
* $ret_4 = $obj->popFront();
* $ret_5 = $obj->popMiddle();
* $ret_6 = $obj->popBack();
*/
```

Dart:

```
class FrontMiddleBackQueue {

FrontMiddleBackQueue() {

}

void pushFront(int val) {
```

```
}

void pushMiddle(int val) {

}

void pushBack(int val) {

}

int popFront() {

}

int popMiddle() {

}

int popBack() {

}
}

/**
* Your FrontMiddleBackQueue object will be instantiated and called as such:
* FrontMiddleBackQueue obj = FrontMiddleBackQueue();
* obj.pushFront(val);
* obj.pushMiddle(val);
* obj.pushBack(val);
* int param4 = obj.popFront();
* int param5 = obj.popMiddle();
* int param6 = obj.popBack();
*/
```

Scala:

```
class FrontMiddleBackQueue() {

def pushFront(`val`: Int): Unit = {

}
```

```
def pushMiddle(`val`: Int): Unit = {

}

def pushBack(`val`: Int): Unit = {

}

def popFront(): Int = {

}

def popMiddle(): Int = {

}

def popBack(): Int = {

}

}

/**
* Your FrontMiddleBackQueue object will be instantiated and called as such:
* val obj = new FrontMiddleBackQueue()
* obj.pushFront(`val`)
* obj.pushMiddle(`val`)
* obj.pushBack(`val`)
* val param_4 = obj.popFront()
* val param_5 = obj.popMiddle()
* val param_6 = obj.popBack()
*/
```

Elixir:

```
defmodule FrontMiddleBackQueue do
@spec init_() :: any
def init_() do

end
```

```
@spec push_front(val :: integer) :: any
def push_front(val) do

end

@spec push_middle(val :: integer) :: any
def push_middle(val) do

end

@spec push_back(val :: integer) :: any
def push_back(val) do

end

@spec pop_front() :: integer
def pop_front() do

end

@spec pop_middle() :: integer
def pop_middle() do

end

@spec pop_back() :: integer
def pop_back() do

end
end

# Your functions will be called as such:
# FrontMiddleBackQueue.init_()
# FrontMiddleBackQueue.push_front(val)
# FrontMiddleBackQueue.push_middle(val)
# FrontMiddleBackQueue.push_back(val)
# param_4 = FrontMiddleBackQueue.pop_front()
# param_5 = FrontMiddleBackQueue.pop_middle()
# param_6 = FrontMiddleBackQueue.pop_back()

# FrontMiddleBackQueue.init_ will be called before every test case, in which
you can do some necessary initializations.
```

Erlang:

```
-spec front_middle_back_queue_init_() -> any().
front_middle_back_queue_init_() ->
.

-spec front_middle_back_queue_push_front(Val :: integer()) -> any().
front_middle_back_queue_push_front(Val) ->
.

-spec front_middle_back_queue_push_middle(Val :: integer()) -> any().
front_middle_back_queue_push_middle(Val) ->
.

-spec front_middle_back_queue_push_back(Val :: integer()) -> any().
front_middle_back_queue_push_back(Val) ->
.

-spec front_middle_back_queue_pop_front() -> integer().
front_middle_back_queue_pop_front() ->
.

-spec front_middle_back_queue_pop_middle() -> integer().
front_middle_back_queue_pop_middle() ->
.

-spec front_middle_back_queue_pop_back() -> integer().
front_middle_back_queue_pop_back() ->
.


%% Your functions will be called as such:
%% front_middle_back_queue_init_(),
%% front_middle_back_queue_push_front(Val),
%% front_middle_back_queue_push_middle(Val),
%% front_middle_back_queue_push_back(Val),
%% Param_4 = front_middle_back_queue_pop_front(),
%% Param_5 = front_middle_back_queue_pop_middle(),
%% Param_6 = front_middle_back_queue_pop_back(),

%% front_middle_back_queue_init_ will be called before every test case, in
which you can do some necessary initializations.
```

Racket:

```
(define front-middle-back-queue%
(class object%
(super-new)

(init-field)

; push-front : exact-integer? -> void?
(define/public (push-front val)
)
; push-middle : exact-integer? -> void?
(define/public (push-middle val)
)
; push-back : exact-integer? -> void?
(define/public (push-back val)
)
; pop-front : -> exact-integer?
(define/public (pop-front)
)
; pop-middle : -> exact-integer?
(define/public (pop-middle)
)
; pop-back : -> exact-integer?
(define/public (pop-back)
)))

;; Your front-middle-back-queue% object will be instantiated and called as
such:
;; (define obj (new front-middle-back-queue%))
;; (send obj push-front val)
;; (send obj push-middle val)
;; (send obj push-back val)
;; (define param_4 (send obj pop-front))
;; (define param_5 (send obj pop-middle))
;; (define param_6 (send obj pop-back))
```

Solutions

C++ Solution:

```
/*
* Problem: Design Front Middle Back Queue
* Difficulty: Medium
* Tags: array, linked_list, queue
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

class FrontMiddleBackQueue {
public:
FrontMiddleBackQueue() {

}

void pushFront(int val) {

}

void pushMiddle(int val) {

}

void pushBack(int val) {

}

int popFront() {

}

int popMiddle() {

}

int popBack() {

}
};

/**
* Your FrontMiddleBackQueue object will be instantiated and called as such:
```

```
* FrontMiddleBackQueue* obj = new FrontMiddleBackQueue();
* obj->pushFront(val);
* obj->pushMiddle(val);
* obj->pushBack(val);
* int param_4 = obj->popFront();
* int param_5 = obj->popMiddle();
* int param_6 = obj->popBack();
*/
```

Java Solution:

```
/**
* Problem: Design Front Middle Back Queue
* Difficulty: Medium
* Tags: array, linked_list, queue
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

class FrontMiddleBackQueue {

public FrontMiddleBackQueue() {

}

public void pushFront(int val) {

}

public void pushMiddle(int val) {

}

public void pushBack(int val) {

}

public int popFront() {
```

```
}

public int popMiddle() {

}

public int popBack() {

}
}

/**
* Your FrontMiddleBackQueue object will be instantiated and called as such:
* FrontMiddleBackQueue obj = new FrontMiddleBackQueue();
* obj.pushFront(val);
* obj.pushMiddle(val);
* obj.pushBack(val);
* int param_4 = obj.popFront();
* int param_5 = obj.popMiddle();
* int param_6 = obj.popBack();
*/
```

Python3 Solution:

```
"""
Problem: Design Front Middle Back Queue
Difficulty: Medium
Tags: array, linked_list, queue

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(1) to O(n) depending on approach
"""

class FrontMiddleBackQueue:

def __init__(self):


def pushFront(self, val: int) -> None:
# TODO: Implement optimized solution
```

```
pass
```

Python Solution:

```
class FrontMiddleBackQueue(object):

def __init__(self):


def pushFront(self, val):
"""
:type val: int
:rtype: None
"""


def pushMiddle(self, val):
"""
:type val: int
:rtype: None
"""


def pushBack(self, val):
"""
:type val: int
:rtype: None
"""


def popFront(self):
"""
:rtype: int
"""


def popMiddle(self):
"""
:rtype: int
"""
```

```
def popBack(self):
"""
:rtype: int
"""



# Your FrontMiddleBackQueue object will be instantiated and called as such:
# obj = FrontMiddleBackQueue()
# obj.pushFront(val)
# obj.pushMiddle(val)
# obj.pushBack(val)
# param_4 = obj.popFront()
# param_5 = obj.popMiddle()
# param_6 = obj.popBack()
```

JavaScript Solution:

```
/**
* Problem: Design Front Middle Back Queue
* Difficulty: Medium
* Tags: array, linked_list, queue
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/


var FrontMiddleBackQueue = function() {

};

/**
* @param {number} val
* @return {void}
*/
FrontMiddleBackQueue.prototype.pushFront = function(val) {

};
```

```
/**
* @param {number} val
* @return {void}
*/
FrontMiddleBackQueue.prototype.pushMiddle = function(val) {

};

/**
* @param {number} val
* @return {void}
*/
FrontMiddleBackQueue.prototype.pushBack = function(val) {

};

/**
* @return {number}
*/
FrontMiddleBackQueue.prototype.popFront = function() {

};

/**
* @return {number}
*/
FrontMiddleBackQueue.prototype.popMiddle = function() {

};

/**
* @return {number}
*/
FrontMiddleBackQueue.prototype.popBack = function() {

};

/**
* Your FrontMiddleBackQueue object will be instantiated and called as such:
* var obj = new FrontMiddleBackQueue()
* obj.pushFront(val)
```

```
* obj.pushMiddle(val)
* obj.pushBack(val)
* var param_4 = obj.popFront()
* var param_5 = obj.popMiddle()
* var param_6 = obj.popBack()
*/
```

TypeScript Solution:

```
/**
* Problem: Design Front Middle Back Queue
* Difficulty: Medium
* Tags: array, linked_list, queue
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

class FrontMiddleBackQueue {
constructor() {

}

pushFront(val: number): void {

}

pushMiddle(val: number): void {

}

pushBack(val: number): void {

}

popFront(): number {

}

popMiddle(): number {
```

```
}

popBack(): number {

}
}

/**
* Your FrontMiddleBackQueue object will be instantiated and called as such:
* var obj = new FrontMiddleBackQueue()
* obj.pushFront(val)
* obj.pushMiddle(val)
* obj.pushBack(val)
* var param_4 = obj.popFront()
* var param_5 = obj.popMiddle()
* var param_6 = obj.popBack()
*/
```

C# Solution:

```
/*
* Problem: Design Front Middle Back Queue
* Difficulty: Medium
* Tags: array, linked_list, queue
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

public class FrontMiddleBackQueue {

public FrontMiddleBackQueue() {

}

public void PushFront(int val) {

}
```

```
public void PushMiddle(int val) {

}

public void PushBack(int val) {

}

public int PopFront() {

}

public int PopMiddle() {

}

public int PopBack() {

}
}

/**
* Your FrontMiddleBackQueue object will be instantiated and called as such:
* FrontMiddleBackQueue obj = new FrontMiddleBackQueue();
* obj.PushFront(val);
* obj.PushMiddle(val);
* obj.PushBack(val);
* int param_4 = obj.PopFront();
* int param_5 = obj.PopMiddle();
* int param_6 = obj.PopBack();
*/
```

C Solution:

```
/*
* Problem: Design Front Middle Back Queue
* Difficulty: Medium
* Tags: array, linked_list, queue
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
```

```
* Space Complexity: O(1) to O(n) depending on approach
*/



typedef struct {

} FrontMiddleBackQueue;


FrontMiddleBackQueue* frontMiddleBackQueueCreate() {

}

void frontMiddleBackQueuePushFront(FrontMiddleBackQueue* obj, int val) {

}

void frontMiddleBackQueuePushMiddle(FrontMiddleBackQueue* obj, int val) {

}

void frontMiddleBackQueuePushBack(FrontMiddleBackQueue* obj, int val) {

}

int frontMiddleBackQueuePopFront(FrontMiddleBackQueue* obj) {

}

int frontMiddleBackQueuePopMiddle(FrontMiddleBackQueue* obj) {

}

int frontMiddleBackQueuePopBack(FrontMiddleBackQueue* obj) {

}

void frontMiddleBackQueueFree(FrontMiddleBackQueue* obj) {

}
```

```
/**
* Your FrontMiddleBackQueue struct will be instantiated and called as such:
* FrontMiddleBackQueue* obj = frontMiddleBackQueueCreate();
* frontMiddleBackQueuePushFront(obj, val);

* frontMiddleBackQueuePushMiddle(obj, val);

* frontMiddleBackQueuePushBack(obj, val);

* int param_4 = frontMiddleBackQueuePopFront(obj);

* int param_5 = frontMiddleBackQueuePopMiddle(obj);

* int param_6 = frontMiddleBackQueuePopBack(obj);

* frontMiddleBackQueueFree(obj);
*/
```

Go Solution:

```
// Problem: Design Front Middle Back Queue
// Difficulty: Medium
// Tags: array, linked_list, queue
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(1) to O(n) depending on approach

type FrontMiddleBackQueue struct {

}


func Constructor() FrontMiddleBackQueue {

}


func (this *FrontMiddleBackQueue) PushFront(val int) {
```

```
}


func (this *FrontMiddleBackQueue) PushMiddle(val int) {

}


func (this *FrontMiddleBackQueue) PushBack(val int) {

}


func (this *FrontMiddleBackQueue) PopFront() int {

}


func (this *FrontMiddleBackQueue) PopMiddle() int {

}


func (this *FrontMiddleBackQueue) PopBack() int {

}


/**
* Your FrontMiddleBackQueue object will be instantiated and called as such:
* obj := Constructor();
* obj.PushFront(val);
* obj.PushMiddle(val);
* obj.PushBack(val);
* param_4 := obj.PopFront();
* param_5 := obj.PopMiddle();
* param_6 := obj.PopBack();
*/
```

Kotlin Solution:

```
class FrontMiddleBackQueue() {

fun pushFront(`val`: Int) {

}

fun pushMiddle(`val`: Int) {

}

fun pushBack(`val`: Int) {

}

fun popFront(): Int {

}

fun popMiddle(): Int {

}

fun popBack(): Int {

}

}

/**
* Your FrontMiddleBackQueue object will be instantiated and called as such:
* var obj = FrontMiddleBackQueue()
* obj.pushFront(`val`)
* obj.pushMiddle(`val`)
* obj.pushBack(`val`)
* var param_4 = obj.popFront()
* var param_5 = obj.popMiddle()
* var param_6 = obj.popBack()
*/
```

Swift Solution:

```
class FrontMiddleBackQueue {

init() {

}

func pushFront(_ val: Int) {

}

func pushMiddle(_ val: Int) {

}

func pushBack(_ val: Int) {

}

func popFront() -> Int {

}

func popMiddle() -> Int {

}

func popBack() -> Int {

}
}

/**
* Your FrontMiddleBackQueue object will be instantiated and called as such:
* let obj = FrontMiddleBackQueue()
* obj.pushFront(val)
* obj.pushMiddle(val)
* obj.pushBack(val)
* let ret_4: Int = obj.popFront()
* let ret_5: Int = obj.popMiddle()
* let ret_6: Int = obj.popBack()
*/
```

Rust Solution:

```
// Problem: Design Front Middle Back Queue
// Difficulty: Medium
// Tags: array, linked_list, queue
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(1) to O(n) depending on approach

struct FrontMiddleBackQueue {

}



/**
* `&self` means the method takes an immutable reference.
* If you need a mutable reference, change it to `&mut self` instead.
*/
impl FrontMiddleBackQueue {

fn new() -> Self {

}

fn push_front(&self, val: i32) {

}

fn push_middle(&self, val: i32) {

}

fn push_back(&self, val: i32) {

}

fn pop_front(&self) -> i32 {

}

fn pop_middle(&self) -> i32 {
```

```
}

fn pop_back(&self) -> i32 {

}
}

/**
* Your FrontMiddleBackQueue object will be instantiated and called as such:
* let obj = FrontMiddleBackQueue::new();
* obj.push_front(val);
* obj.push_middle(val);
* obj.push_back(val);
* let ret_4: i32 = obj.pop_front();
* let ret_5: i32 = obj.pop_middle();
* let ret_6: i32 = obj.pop_back();
*/
```

Ruby Solution:

```
class FrontMiddleBackQueue
def initialize()

end


=begin
:type val: Integer
:rtype: Void
=end
def push_front(val)

end


=begin
:type val: Integer
:rtype: Void
=end
def push_middle(val)
```

```
end


=begin
:type val: Integer
:rtype: Void
=end
def push_back(val)

end


=begin
:rtype: Integer
=end
def pop_front()

end


=begin
:rtype: Integer
=end
def pop_middle()

end


=begin
:rtype: Integer
=end
def pop_back()

end


end

# Your FrontMiddleBackQueue object will be instantiated and called as such:
# obj = FrontMiddleBackQueue.new()
# obj.push_front(val)
```

```
# obj.push_middle(val)
# obj.push_back(val)
# param_4 = obj.pop_front()
# param_5 = obj.pop_middle()
# param_6 = obj.pop_back()
```

PHP Solution:

```
class FrontMiddleBackQueue {
/**
*/
function __construct() {

}

/**
* @param Integer $val
* @return NULL
*/
function pushFront($val) {

}

/**
* @param Integer $val
* @return NULL
*/
function pushMiddle($val) {

}

/**
* @param Integer $val
* @return NULL
*/
function pushBack($val) {

}

/**
* @return Integer
```

```
*/
function popFront() {

}

/**
* @return Integer
*/
function popMiddle() {

}

/**
* @return Integer
*/
function popBack() {

}
}

/**
* Your FrontMiddleBackQueue object will be instantiated and called as such:
* $obj = FrontMiddleBackQueue();
* $obj->pushFront($val);
* $obj->pushMiddle($val);
* $obj->pushBack($val);
* $ret_4 = $obj->popFront();
* $ret_5 = $obj->popMiddle();
* $ret_6 = $obj->popBack();
*/
```

Dart Solution:

```
class FrontMiddleBackQueue {

FrontMiddleBackQueue() {

}

void pushFront(int val) {
```

```
}

void pushMiddle(int val) {

}

void pushBack(int val) {

}

int popFront() {

}

int popMiddle() {

}

int popBack() {

}
}

/**
* Your FrontMiddleBackQueue object will be instantiated and called as such:
* FrontMiddleBackQueue obj = FrontMiddleBackQueue();
* obj.pushFront(val);
* obj.pushMiddle(val);
* obj.pushBack(val);
* int param4 = obj.popFront();
* int param5 = obj.popMiddle();
* int param6 = obj.popBack();
*/
```

Scala Solution:

```
class FrontMiddleBackQueue() {

def pushFront(`val`: Int): Unit = {

}
```

```
def pushMiddle(`val`: Int): Unit = {

}

def pushBack(`val`: Int): Unit = {

}

def popFront(): Int = {

}

def popMiddle(): Int = {

}

def popBack(): Int = {

}

}

/**
* Your FrontMiddleBackQueue object will be instantiated and called as such:
* val obj = new FrontMiddleBackQueue()
* obj.pushFront(`val`)
* obj.pushMiddle(`val`)
* obj.pushBack(`val`)
* val param_4 = obj.popFront()
* val param_5 = obj.popMiddle()
* val param_6 = obj.popBack()
*/
```

Elixir Solution:

```
defmodule FrontMiddleBackQueue do
@spec init_() :: any
def init_() do

end
```

```
@spec push_front(val :: integer) :: any
def push_front(val) do

end

@spec push_middle(val :: integer) :: any
def push_middle(val) do

end

@spec push_back(val :: integer) :: any
def push_back(val) do

end

@spec pop_front() :: integer
def pop_front() do

end

@spec pop_middle() :: integer
def pop_middle() do

end

@spec pop_back() :: integer
def pop_back() do

end
end

# Your functions will be called as such:
# FrontMiddleBackQueue.init_()
# FrontMiddleBackQueue.push_front(val)
# FrontMiddleBackQueue.push_middle(val)
# FrontMiddleBackQueue.push_back(val)
# param_4 = FrontMiddleBackQueue.pop_front()
# param_5 = FrontMiddleBackQueue.pop_middle()
# param_6 = FrontMiddleBackQueue.pop_back()

# FrontMiddleBackQueue.init_ will be called before every test case, in which
```

```
you can do some necessary initializations.
```

Erlang Solution:

```
-spec front_middle_back_queue_init_() -> any().
front_middle_back_queue_init_() ->
.

-spec front_middle_back_queue_push_front(Val :: integer()) -> any().
front_middle_back_queue_push_front(Val) ->
.

-spec front_middle_back_queue_push_middle(Val :: integer()) -> any().
front_middle_back_queue_push_middle(Val) ->
.

-spec front_middle_back_queue_push_back(Val :: integer()) -> any().
front_middle_back_queue_push_back(Val) ->
.

-spec front_middle_back_queue_pop_front() -> integer().
front_middle_back_queue_pop_front() ->
.

-spec front_middle_back_queue_pop_middle() -> integer().
front_middle_back_queue_pop_middle() ->
.

-spec front_middle_back_queue_pop_back() -> integer().
front_middle_back_queue_pop_back() ->
.


%% Your functions will be called as such:
%% front_middle_back_queue_init_(),
%% front_middle_back_queue_push_front(Val),
%% front_middle_back_queue_push_middle(Val),
%% front_middle_back_queue_push_back(Val),
%% Param_4 = front_middle_back_queue_pop_front(),
%% Param_5 = front_middle_back_queue_pop_middle(),
%% Param_6 = front_middle_back_queue_pop_back(),
```

```
%% front_middle_back_queue_init_ will be called before every test case, in
which you can do some necessary initializations.
```

Racket Solution:

```
(define front-middle-back-queue%
(class object%
(super-new)

(init-field)

; push-front : exact-integer? -> void?
(define/public (push-front val)
)
; push-middle : exact-integer? -> void?
(define/public (push-middle val)
)
; push-back : exact-integer? -> void?
(define/public (push-back val)
)
; pop-front : -> exact-integer?
(define/public (pop-front)
)
; pop-middle : -> exact-integer?
(define/public (pop-middle)
)
; pop-back : -> exact-integer?
(define/public (pop-back)
)))

;; Your front-middle-back-queue% object will be instantiated and called as
such:
;; (define obj (new front-middle-back-queue%))
;; (send obj push-front val)
;; (send obj push-middle val)
;; (send obj push-back val)
;; (define param_4 (send obj pop-front))
;; (define param_5 (send obj pop-middle))
;; (define param_6 (send obj pop-back))
```