Problem 641: Design Circular Deque

Problem Information

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

Problem Description

Design your implementation of the circular double-ended queue (deque).

Implement the

MyCircularDeque

class:

MyCircularDeque(int k)

Initializes the deque with a maximum size of

k

.

boolean insertFront()

Adds an item at the front of Deque. Returns

true

if the operation is successful, or

false

otherwise.

boolean insertLast()

Adds an item at the rear of Deque. Returns

true

if the operation is successful, or

false

otherwise.

boolean deleteFront()

Deletes an item from the front of Deque. Returns

true

if the operation is successful, or

false

otherwise.

boolean deleteLast()

Deletes an item from the rear of Deque. Returns

true

if the operation is successful, or

false

otherwise.

int getFront()

Returns the front item from the Deque. Returns

-1

if the deque is empty.

int getRear()

Returns the last item from Deque. Returns

-1

if the deque is empty.

boolean isEmpty()

Returns

true

if the deque is empty, or

false

otherwise.

boolean isFull()

Returns

true

if the deque is full, or

false

otherwise.

Example 1:

Input

["MyCircularDeque", "insertLast", "insertLast", "insertFront", "insertFront", "getRear", "isFull", "deleteLast", "insertFront", "getFront"] [[3], [1], [2], [3], [4], [], [], [], [4], []]

Output

[null, true, true, true, false, 2, true, true, true, 4]

Explanation

MyCircularDeque myCircularDeque = new MyCircularDeque(3); myCircularDeque.insertLast(1); // return True myCircularDeque.insertLast(2); // return True myCircularDeque.insertFront(3); // return True myCircularDeque.insertFront(4); // return False, the queue is full. myCircularDeque.getRear(); // return 2 myCircularDeque.isFull(); // return True myCircularDeque.deleteLast(); // return True myCircularDeque.insertFront(4); // return True myCircularDeque.getFront(); // return 4

Constraints:

1 <= k <= 1000

0 <= value <= 1000

At most

2000

calls will be made to

insertFront

,

insertLast

,

deleteFront

,

deleteLast

,

getFront

,

getRear

,

isEmpty

,

isFull

.

Code Snippets

C++:

```
class MyCircularDeque {
public:
MyCircularDeque(int k) {

}

bool insertFront(int value) {

}
```

```
bool insertLast(int value) {

}

bool deleteFront() {

}

bool deleteLast() {

}

int getFront() {

}

int getRear() {

}

bool isEmpty() {

}

bool isFull() {

}
};

/**
* Your MyCircularDeque object will be instantiated and called as such:
* MyCircularDeque* obj = new MyCircularDeque(k);
* bool param_1 = obj->insertFront(value);
* bool param_2 = obj->insertLast(value);
* bool param_3 = obj->deleteFront();
* bool param_4 = obj->deleteLast();
* int param_5 = obj->getFront();
* int param_6 = obj->getRear();
* bool param_7 = obj->isEmpty();
* bool param_8 = obj->isFull();
*/
```

Java:

```
class MyCircularDeque {

public MyCircularDeque(int k) {

}

public boolean insertFront(int value) {

}

public boolean insertLast(int value) {

}

public boolean deleteFront() {

}

public boolean deleteLast() {

}

public int getFront() {

}

public int getRear() {

}

public boolean isEmpty() {

}

public boolean isFull() {

}
}

/**
```

```
* Your MyCircularDeque object will be instantiated and called as such:
* MyCircularDeque obj = new MyCircularDeque(k);
* boolean param_1 = obj.insertFront(value);
* boolean param_2 = obj.insertLast(value);
* boolean param_3 = obj.deleteFront();
* boolean param_4 = obj.deleteLast();
* int param_5 = obj.getFront();
* int param_6 = obj.getRear();
* boolean param_7 = obj.isEmpty();
* boolean param_8 = obj.isFull();
*/
```

Python3:

```
class MyCircularDeque:

def __init__(self, k: int):


def insertFront(self, value: int) -> bool:


def insertLast(self, value: int) -> bool:


def deleteFront(self) -> bool:


def deleteLast(self) -> bool:


def getFront(self) -> int:


def getRear(self) -> int:


def isEmpty(self) -> bool:


def isFull(self) -> bool:
```

```
# Your MyCircularDeque object will be instantiated and called as such:
# obj = MyCircularDeque(k)
# param_1 = obj.insertFront(value)
# param_2 = obj.insertLast(value)
# param_3 = obj.deleteFront()
# param_4 = obj.deleteLast()
# param_5 = obj.getFront()
# param_6 = obj.getRear()
# param_7 = obj.isEmpty()
# param_8 = obj.isFull()
```

Python:

```
class MyCircularDeque(object):

def __init__(self, k):
"""
:type k: int
"""


def insertFront(self, value):
"""
:type value: int
:rtype: bool
"""


def insertLast(self, value):
"""
:type value: int
:rtype: bool
"""


def deleteFront(self):
"""
:rtype: bool
"""
```

```
def deleteLast(self):
"""
:rtype: bool
"""


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


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


def isEmpty(self):
"""
:rtype: bool
"""


def isFull(self):
"""
:rtype: bool
"""



# Your MyCircularDeque object will be instantiated and called as such:
# obj = MyCircularDeque(k)
# param_1 = obj.insertFront(value)
# param_2 = obj.insertLast(value)
# param_3 = obj.deleteFront()
# param_4 = obj.deleteLast()
# param_5 = obj.getFront()
# param_6 = obj.getRear()
# param_7 = obj.isEmpty()
# param_8 = obj.isFull()
```

JavaScript:

```
/**
* @param {number} k
*/
var MyCircularDeque = function(k) {

};

/**
* @param {number} value
* @return {boolean}
*/
MyCircularDeque.prototype.insertFront = function(value) {

};

/**
* @param {number} value
* @return {boolean}
*/
MyCircularDeque.prototype.insertLast = function(value) {

};

/**
* @return {boolean}
*/
MyCircularDeque.prototype.deleteFront = function() {

};

/**
* @return {boolean}
*/
MyCircularDeque.prototype.deleteLast = function() {

};

/**
* @return {number}
*/
```

```
MyCircularDeque.prototype.getFront = function() {

};

/**
* @return {number}
*/
MyCircularDeque.prototype.getRear = function() {

};

/**
* @return {boolean}
*/
MyCircularDeque.prototype.isEmpty = function() {

};

/**
* @return {boolean}
*/
MyCircularDeque.prototype.isFull = function() {

};

/**
* Your MyCircularDeque object will be instantiated and called as such:
* var obj = new MyCircularDeque(k)
* var param_1 = obj.insertFront(value)
* var param_2 = obj.insertLast(value)
* var param_3 = obj.deleteFront()
* var param_4 = obj.deleteLast()
* var param_5 = obj.getFront()
* var param_6 = obj.getRear()
* var param_7 = obj.isEmpty()
* var param_8 = obj.isFull()
*/
```

TypeScript:

```
class MyCircularDeque {
constructor(k: number) {
```

```
}

insertFront(value: number): boolean {

}

insertLast(value: number): boolean {

}

deleteFront(): boolean {

}

deleteLast(): boolean {

}

getFront(): number {

}

getRear(): number {

}

isEmpty(): boolean {

}

isFull(): boolean {

}
}

/**
* Your MyCircularDeque object will be instantiated and called as such:
* var obj = new MyCircularDeque(k)
* var param_1 = obj.insertFront(value)
* var param_2 = obj.insertLast(value)
* var param_3 = obj.deleteFront()
```

```
* var param_4 = obj.deleteLast()
* var param_5 = obj.getFront()
* var param_6 = obj.getRear()
* var param_7 = obj.isEmpty()
* var param_8 = obj.isFull()
*/
```

C#:

```
public class MyCircularDeque {

public MyCircularDeque(int k) {

}

public bool InsertFront(int value) {

}

public bool InsertLast(int value) {

}

public bool DeleteFront() {

}

public bool DeleteLast() {

}

public int GetFront() {

}

public int GetRear() {

}

public bool IsEmpty() {

}
```

```
public bool IsFull() {

}
}

/**
* Your MyCircularDeque object will be instantiated and called as such:
* MyCircularDeque obj = new MyCircularDeque(k);
* bool param_1 = obj.InsertFront(value);
* bool param_2 = obj.InsertLast(value);
* bool param_3 = obj.DeleteFront();
* bool param_4 = obj.DeleteLast();
* int param_5 = obj.GetFront();
* int param_6 = obj.GetRear();
* bool param_7 = obj.IsEmpty();
* bool param_8 = obj.IsFull();
*/
```

C:

```
typedef struct {

} MyCircularDeque;


MyCircularDeque* myCircularDequeCreate(int k) {

}

bool myCircularDequeInsertFront(MyCircularDeque* obj, int value) {

}

bool myCircularDequeInsertLast(MyCircularDeque* obj, int value) {

}

bool myCircularDequeDeleteFront(MyCircularDeque* obj) {
```

```
}

bool myCircularDequeDeleteLast(MyCircularDeque* obj) {

}

int myCircularDequeGetFront(MyCircularDeque* obj) {

}

int myCircularDequeGetRear(MyCircularDeque* obj) {

}

bool myCircularDequeIsEmpty(MyCircularDeque* obj) {

}

bool myCircularDequeIsFull(MyCircularDeque* obj) {

}

void myCircularDequeFree(MyCircularDeque* obj) {

}

/**
* Your MyCircularDeque struct will be instantiated and called as such:
* MyCircularDeque* obj = myCircularDequeCreate(k);
* bool param_1 = myCircularDequeInsertFront(obj, value);

* bool param_2 = myCircularDequeInsertLast(obj, value);

* bool param_3 = myCircularDequeDeleteFront(obj);

* bool param_4 = myCircularDequeDeleteLast(obj);

* int param_5 = myCircularDequeGetFront(obj);

* int param_6 = myCircularDequeGetRear(obj);
```

```
* bool param_7 = myCircularDequeIsEmpty(obj);

* bool param_8 = myCircularDequeIsFull(obj);

* myCircularDequeFree(obj);
*/
```

Go:

```
type MyCircularDeque struct {

}


func Constructor(k int) MyCircularDeque {

}


func (this *MyCircularDeque) InsertFront(value int) bool {

}


func (this *MyCircularDeque) InsertLast(value int) bool {

}


func (this *MyCircularDeque) DeleteFront() bool {

}


func (this *MyCircularDeque) DeleteLast() bool {

}


func (this *MyCircularDeque) GetFront() int {

}
```

```
func (this *MyCircularDeque) GetRear() int {

}


func (this *MyCircularDeque) IsEmpty() bool {

}


func (this *MyCircularDeque) IsFull() bool {

}


/**
* Your MyCircularDeque object will be instantiated and called as such:
* obj := Constructor(k);
* param_1 := obj.InsertFront(value);
* param_2 := obj.InsertLast(value);
* param_3 := obj.DeleteFront();
* param_4 := obj.DeleteLast();
* param_5 := obj.GetFront();
* param_6 := obj.GetRear();
* param_7 := obj.IsEmpty();
* param_8 := obj.IsFull();
*/
```

Kotlin:

```
class MyCircularDeque(k: Int) {

fun insertFront(value: Int): Boolean {

}

fun insertLast(value: Int): Boolean {

}
```

```
fun deleteFront(): Boolean {

}

fun deleteLast(): Boolean {

}

fun getFront(): Int {

}

fun getRear(): Int {

}

fun isEmpty(): Boolean {

}

fun isFull(): Boolean {

}

}

/**
* Your MyCircularDeque object will be instantiated and called as such:
* var obj = MyCircularDeque(k)
* var param_1 = obj.insertFront(value)
* var param_2 = obj.insertLast(value)
* var param_3 = obj.deleteFront()
* var param_4 = obj.deleteLast()
* var param_5 = obj.getFront()
* var param_6 = obj.getRear()
* var param_7 = obj.isEmpty()
* var param_8 = obj.isFull()
*/
```

Swift:

```
class MyCircularDeque {

init(_ k: Int) {

}

func insertFront(_ value: Int) -> Bool {

}

func insertLast(_ value: Int) -> Bool {

}

func deleteFront() -> Bool {

}

func deleteLast() -> Bool {

}

func getFront() -> Int {

}

func getRear() -> Int {

}

func isEmpty() -> Bool {

}

func isFull() -> Bool {

}
}

/**
* Your MyCircularDeque object will be instantiated and called as such:
* let obj = MyCircularDeque(k)
```

```
* let ret_1: Bool = obj.insertFront(value)
* let ret_2: Bool = obj.insertLast(value)
* let ret_3: Bool = obj.deleteFront()
* let ret_4: Bool = obj.deleteLast()
* let ret_5: Int = obj.getFront()
* let ret_6: Int = obj.getRear()
* let ret_7: Bool = obj.isEmpty()
* let ret_8: Bool = obj.isFull()
*/
```

Rust:

```
struct MyCircularDeque {

}


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

fn new(k: i32) -> Self {

}

fn insert_front(&self, value: i32) -> bool {

}

fn insert_last(&self, value: i32) -> bool {

}

fn delete_front(&self) -> bool {

}

fn delete_last(&self) -> bool {

}
```

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

}

fn get_rear(&self) -> i32 {

}

fn is_empty(&self) -> bool {

}

fn is_full(&self) -> bool {

}
}

/**
* Your MyCircularDeque object will be instantiated and called as such:
* let obj = MyCircularDeque::new(k);
* let ret_1: bool = obj.insert_front(value);
* let ret_2: bool = obj.insert_last(value);
* let ret_3: bool = obj.delete_front();
* let ret_4: bool = obj.delete_last();
* let ret_5: i32 = obj.get_front();
* let ret_6: i32 = obj.get_rear();
* let ret_7: bool = obj.is_empty();
* let ret_8: bool = obj.is_full();
*/
```

Ruby:

```
class MyCircularDeque

=begin
:type k: Integer
=end
def initialize(k)

end
```

```
=begin
:type value: Integer
:rtype: Boolean
=end
def insert_front(value)

end


=begin
:type value: Integer
:rtype: Boolean
=end
def insert_last(value)

end


=begin
:rtype: Boolean
=end
def delete_front()

end


=begin
:rtype: Boolean
=end
def delete_last()

end


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

end
```

```
=begin
:rtype: Integer
=end
def get_rear()

end


=begin
:rtype: Boolean
=end
def is_empty()

end


=begin
:rtype: Boolean
=end
def is_full()

end


end

# Your MyCircularDeque object will be instantiated and called as such:
# obj = MyCircularDeque.new(k)
# param_1 = obj.insert_front(value)
# param_2 = obj.insert_last(value)
# param_3 = obj.delete_front()
# param_4 = obj.delete_last()
# param_5 = obj.get_front()
# param_6 = obj.get_rear()
# param_7 = obj.is_empty()
# param_8 = obj.is_full()
```

PHP:

```
class MyCircularDeque {
/**
```

```
* @param Integer $k
*/
function __construct($k) {

}

/**
* @param Integer $value
* @return Boolean
*/
function insertFront($value) {

}

/**
* @param Integer $value
* @return Boolean
*/
function insertLast($value) {

}

/**
* @return Boolean
*/
function deleteFront() {

}

/**
* @return Boolean
*/
function deleteLast() {

}

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

}
```

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

}

/**
* @return Boolean
*/
function isEmpty() {

}

/**
* @return Boolean
*/
function isFull() {

}
}

/**
* Your MyCircularDeque object will be instantiated and called as such:
* $obj = MyCircularDeque($k);
* $ret_1 = $obj->insertFront($value);
* $ret_2 = $obj->insertLast($value);
* $ret_3 = $obj->deleteFront();
* $ret_4 = $obj->deleteLast();
* $ret_5 = $obj->getFront();
* $ret_6 = $obj->getRear();
* $ret_7 = $obj->isEmpty();
* $ret_8 = $obj->isFull();
*/
```

Dart:

```
class MyCircularDeque {

MyCircularDeque(int k) {
```

```
}

bool insertFront(int value) {

}

bool insertLast(int value) {

}

bool deleteFront() {

}

bool deleteLast() {

}

int getFront() {

}

int getRear() {

}

bool isEmpty() {

}

bool isFull() {

}
}

/**
* Your MyCircularDeque object will be instantiated and called as such:
* MyCircularDeque obj = MyCircularDeque(k);
* bool param1 = obj.insertFront(value);
* bool param2 = obj.insertLast(value);
* bool param3 = obj.deleteFront();
* bool param4 = obj.deleteLast();
```

```
* int param5 = obj.getFront();
* int param6 = obj.getRear();
* bool param7 = obj.isEmpty();
* bool param8 = obj.isFull();
*/
```

Scala:

```
class MyCircularDeque(_k: Int) {

def insertFront(value: Int): Boolean = {

}

def insertLast(value: Int): Boolean = {

}

def deleteFront(): Boolean = {

}

def deleteLast(): Boolean = {

}

def getFront(): Int = {

}

def getRear(): Int = {

}

def isEmpty(): Boolean = {

}

def isFull(): Boolean = {

}
```

```
}

/**
* Your MyCircularDeque object will be instantiated and called as such:
* val obj = new MyCircularDeque(k)
* val param_1 = obj.insertFront(value)
* val param_2 = obj.insertLast(value)
* val param_3 = obj.deleteFront()
* val param_4 = obj.deleteLast()
* val param_5 = obj.getFront()
* val param_6 = obj.getRear()
* val param_7 = obj.isEmpty()
* val param_8 = obj.isFull()
*/
```

Elixir:

```
defmodule MyCircularDeque do
@spec init_(k :: integer) :: any
def init_(k) do

end

@spec insert_front(value :: integer) :: boolean
def insert_front(value) do

end

@spec insert_last(value :: integer) :: boolean
def insert_last(value) do

end

@spec delete_front() :: boolean
def delete_front() do

end

@spec delete_last() :: boolean
def delete_last() do

end
```

```
@spec get_front() :: integer
def get_front() do

end

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

end

@spec is_empty() :: boolean
def is_empty() do

end

@spec is_full() :: boolean
def is_full() do

end
end

# Your functions will be called as such:
# MyCircularDeque.init_(k)
# param_1 = MyCircularDeque.insert_front(value)
# param_2 = MyCircularDeque.insert_last(value)
# param_3 = MyCircularDeque.delete_front()
# param_4 = MyCircularDeque.delete_last()
# param_5 = MyCircularDeque.get_front()
# param_6 = MyCircularDeque.get_rear()
# param_7 = MyCircularDeque.is_empty()
# param_8 = MyCircularDeque.is_full()

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

Erlang:

```
-spec my_circular_deque_init_(K :: integer()) -> any().
my_circular_deque_init_(K) ->
.
```

```
-spec my_circular_deque_insert_front(Value :: integer()) -> boolean().
my_circular_deque_insert_front(Value) ->
.

-spec my_circular_deque_insert_last(Value :: integer()) -> boolean().
my_circular_deque_insert_last(Value) ->
.

-spec my_circular_deque_delete_front() -> boolean().
my_circular_deque_delete_front() ->
.

-spec my_circular_deque_delete_last() -> boolean().
my_circular_deque_delete_last() ->
.

-spec my_circular_deque_get_front() -> integer().
my_circular_deque_get_front() ->
.

-spec my_circular_deque_get_rear() -> integer().
my_circular_deque_get_rear() ->
.

-spec my_circular_deque_is_empty() -> boolean().
my_circular_deque_is_empty() ->
.

-spec my_circular_deque_is_full() -> boolean().
my_circular_deque_is_full() ->
.


%% Your functions will be called as such:
%% my_circular_deque_init_(K),
%% Param_1 = my_circular_deque_insert_front(Value),
%% Param_2 = my_circular_deque_insert_last(Value),
%% Param_3 = my_circular_deque_delete_front(),
%% Param_4 = my_circular_deque_delete_last(),
%% Param_5 = my_circular_deque_get_front(),
%% Param_6 = my_circular_deque_get_rear(),
%% Param_7 = my_circular_deque_is_empty(),
```

```
%% Param_8 = my_circular_deque_is_full(),

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

Racket:

```
(define my-circular-deque%
(class object%
(super-new)

; k : exact-integer?
(init-field
k)

; insert-front : exact-integer? -> boolean?
(define/public (insert-front value)
)
; insert-last : exact-integer? -> boolean?
(define/public (insert-last value)
)
; delete-front : -> boolean?
(define/public (delete-front)
)
; delete-last : -> boolean?
(define/public (delete-last)
)
; get-front : -> exact-integer?
(define/public (get-front)
)
; get-rear : -> exact-integer?
(define/public (get-rear)
)
; is-empty : -> boolean?
(define/public (is-empty)
)
; is-full : -> boolean?
(define/public (is-full)
)))

;; Your my-circular-deque% object will be instantiated and called as such:
;; (define obj (new my-circular-deque% [k k]))
```

```
;; (define param_1 (send obj insert-front value))
;; (define param_2 (send obj insert-last value))
;; (define param_3 (send obj delete-front))
;; (define param_4 (send obj delete-last))
;; (define param_5 (send obj get-front))
;; (define param_6 (send obj get-rear))
;; (define param_7 (send obj is-empty))
;; (define param_8 (send obj is-full))
```

Solutions

C++ Solution:

```
/*
* Problem: Design Circular Deque
* 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 MyCircularDeque {
public:
MyCircularDeque(int k) {

}

bool insertFront(int value) {

}

bool insertLast(int value) {

}

bool deleteFront() {

}
```

```
bool deleteLast() {

}

int getFront() {

}

int getRear() {

}

bool isEmpty() {

}

bool isFull() {

}
};

/**
* Your MyCircularDeque object will be instantiated and called as such:
* MyCircularDeque* obj = new MyCircularDeque(k);
* bool param_1 = obj->insertFront(value);
* bool param_2 = obj->insertLast(value);
* bool param_3 = obj->deleteFront();
* bool param_4 = obj->deleteLast();
* int param_5 = obj->getFront();
* int param_6 = obj->getRear();
* bool param_7 = obj->isEmpty();
* bool param_8 = obj->isFull();
*/
```

Java Solution:

```
/**
* Problem: Design Circular Deque
* 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 MyCircularDeque {

public MyCircularDeque(int k) {

}

public boolean insertFront(int value) {

}

public boolean insertLast(int value) {

}

public boolean deleteFront() {

}

public boolean deleteLast() {

}

public int getFront() {

}

public int getRear() {

}

public boolean isEmpty() {

}

public boolean isFull() {
```

```
}
}

/**
* Your MyCircularDeque object will be instantiated and called as such:
* MyCircularDeque obj = new MyCircularDeque(k);
* boolean param_1 = obj.insertFront(value);
* boolean param_2 = obj.insertLast(value);
* boolean param_3 = obj.deleteFront();
* boolean param_4 = obj.deleteLast();
* int param_5 = obj.getFront();
* int param_6 = obj.getRear();
* boolean param_7 = obj.isEmpty();
* boolean param_8 = obj.isFull();
*/
```

Python3 Solution:

```
"""
Problem: Design Circular Deque
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 MyCircularDeque:

def __init__(self, k: int):


def insertFront(self, value: int) -> bool:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class MyCircularDeque(object):
```

```
def __init__(self, k):
"""
:type k: int
"""


def insertFront(self, value):
"""
:type value: int
:rtype: bool
"""


def insertLast(self, value):
"""
:type value: int
:rtype: bool
"""


def deleteFront(self):
"""
:rtype: bool
"""


def deleteLast(self):
"""
:rtype: bool
"""


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


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

```
def isEmpty(self):
"""
:rtype: bool
"""


def isFull(self):
"""
:rtype: bool
"""



# Your MyCircularDeque object will be instantiated and called as such:
# obj = MyCircularDeque(k)
# param_1 = obj.insertFront(value)
# param_2 = obj.insertLast(value)
# param_3 = obj.deleteFront()
# param_4 = obj.deleteLast()
# param_5 = obj.getFront()
# param_6 = obj.getRear()
# param_7 = obj.isEmpty()
# param_8 = obj.isFull()
```

JavaScript Solution:

```
/**
* Problem: Design Circular Deque
* 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
*/

/**
* @param {number} k
*/
```

```
var MyCircularDeque = function(k) {

};

/**
* @param {number} value
* @return {boolean}
*/
MyCircularDeque.prototype.insertFront = function(value) {

};

/**
* @param {number} value
* @return {boolean}
*/
MyCircularDeque.prototype.insertLast = function(value) {

};

/**
* @return {boolean}
*/
MyCircularDeque.prototype.deleteFront = function() {

};

/**
* @return {boolean}
*/
MyCircularDeque.prototype.deleteLast = function() {

};

/**
* @return {number}
*/
MyCircularDeque.prototype.getFront = function() {

};

/**
```

```
* @return {number}
*/
MyCircularDeque.prototype.getRear = function() {

};

/**
* @return {boolean}
*/
MyCircularDeque.prototype.isEmpty = function() {

};

/**
* @return {boolean}
*/
MyCircularDeque.prototype.isFull = function() {

};

/**
* Your MyCircularDeque object will be instantiated and called as such:
* var obj = new MyCircularDeque(k)
* var param_1 = obj.insertFront(value)
* var param_2 = obj.insertLast(value)
* var param_3 = obj.deleteFront()
* var param_4 = obj.deleteLast()
* var param_5 = obj.getFront()
* var param_6 = obj.getRear()
* var param_7 = obj.isEmpty()
* var param_8 = obj.isFull()
*/
```

TypeScript Solution:

```
/**
* Problem: Design Circular Deque
* 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 MyCircularDeque {
constructor(k: number) {

}

insertFront(value: number): boolean {

}

insertLast(value: number): boolean {

}

deleteFront(): boolean {

}

deleteLast(): boolean {

}

getFront(): number {

}

getRear(): number {

}

isEmpty(): boolean {

}

isFull(): boolean {

}
}
```

```
/**
* Your MyCircularDeque object will be instantiated and called as such:
* var obj = new MyCircularDeque(k)
* var param_1 = obj.insertFront(value)
* var param_2 = obj.insertLast(value)
* var param_3 = obj.deleteFront()
* var param_4 = obj.deleteLast()
* var param_5 = obj.getFront()
* var param_6 = obj.getRear()
* var param_7 = obj.isEmpty()
* var param_8 = obj.isFull()
*/
```

C# Solution:

```
/*
* Problem: Design Circular Deque
* 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 MyCircularDeque {

public MyCircularDeque(int k) {

}

public bool InsertFront(int value) {

}

public bool InsertLast(int value) {

}

public bool DeleteFront() {
```

```
}

public bool DeleteLast() {

}

public int GetFront() {

}

public int GetRear() {

}

public bool IsEmpty() {

}

public bool IsFull() {

}
}

/**
* Your MyCircularDeque object will be instantiated and called as such:
* MyCircularDeque obj = new MyCircularDeque(k);
* bool param_1 = obj.InsertFront(value);
* bool param_2 = obj.InsertLast(value);
* bool param_3 = obj.DeleteFront();
* bool param_4 = obj.DeleteLast();
* int param_5 = obj.GetFront();
* int param_6 = obj.GetRear();
* bool param_7 = obj.IsEmpty();
* bool param_8 = obj.IsFull();
*/
```

C Solution:

```
/*
* Problem: Design Circular Deque
* 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 {

} MyCircularDeque;


MyCircularDeque* myCircularDequeCreate(int k) {

}

bool myCircularDequeInsertFront(MyCircularDeque* obj, int value) {

}

bool myCircularDequeInsertLast(MyCircularDeque* obj, int value) {

}

bool myCircularDequeDeleteFront(MyCircularDeque* obj) {

}

bool myCircularDequeDeleteLast(MyCircularDeque* obj) {

}

int myCircularDequeGetFront(MyCircularDeque* obj) {

}

int myCircularDequeGetRear(MyCircularDeque* obj) {

}
```

```
bool myCircularDequeIsEmpty(MyCircularDeque* obj) {

}

bool myCircularDequeIsFull(MyCircularDeque* obj) {

}

void myCircularDequeFree(MyCircularDeque* obj) {

}

/**
* Your MyCircularDeque struct will be instantiated and called as such:
* MyCircularDeque* obj = myCircularDequeCreate(k);
* bool param_1 = myCircularDequeInsertFront(obj, value);

* bool param_2 = myCircularDequeInsertLast(obj, value);

* bool param_3 = myCircularDequeDeleteFront(obj);

* bool param_4 = myCircularDequeDeleteLast(obj);

* int param_5 = myCircularDequeGetFront(obj);

* int param_6 = myCircularDequeGetRear(obj);

* bool param_7 = myCircularDequeIsEmpty(obj);

* bool param_8 = myCircularDequeIsFull(obj);

* myCircularDequeFree(obj);
*/
```

Go Solution:

```
// Problem: Design Circular Deque
// 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 MyCircularDeque struct {

}


func Constructor(k int) MyCircularDeque {

}


func (this *MyCircularDeque) InsertFront(value int) bool {

}


func (this *MyCircularDeque) InsertLast(value int) bool {

}


func (this *MyCircularDeque) DeleteFront() bool {

}


func (this *MyCircularDeque) DeleteLast() bool {

}


func (this *MyCircularDeque) GetFront() int {

}


func (this *MyCircularDeque) GetRear() int {

}
```

```
func (this *MyCircularDeque) IsEmpty() bool {

}


func (this *MyCircularDeque) IsFull() bool {

}


/**
* Your MyCircularDeque object will be instantiated and called as such:
* obj := Constructor(k);
* param_1 := obj.InsertFront(value);
* param_2 := obj.InsertLast(value);
* param_3 := obj.DeleteFront();
* param_4 := obj.DeleteLast();
* param_5 := obj.GetFront();
* param_6 := obj.GetRear();
* param_7 := obj.IsEmpty();
* param_8 := obj.IsFull();
*/
```

Kotlin Solution:

```
class MyCircularDeque(k: Int) {

fun insertFront(value: Int): Boolean {

}

fun insertLast(value: Int): Boolean {

}

fun deleteFront(): Boolean {

}
```

```
fun deleteLast(): Boolean {

}

fun getFront(): Int {

}

fun getRear(): Int {

}

fun isEmpty(): Boolean {

}

fun isFull(): Boolean {

}

}

/**
* Your MyCircularDeque object will be instantiated and called as such:
* var obj = MyCircularDeque(k)
* var param_1 = obj.insertFront(value)
* var param_2 = obj.insertLast(value)
* var param_3 = obj.deleteFront()
* var param_4 = obj.deleteLast()
* var param_5 = obj.getFront()
* var param_6 = obj.getRear()
* var param_7 = obj.isEmpty()
* var param_8 = obj.isFull()
*/
```

Swift Solution:

```
class MyCircularDeque {

init(_ k: Int) {
```

```
}

func insertFront(_ value: Int) -> Bool {

}

func insertLast(_ value: Int) -> Bool {

}

func deleteFront() -> Bool {

}

func deleteLast() -> Bool {

}

func getFront() -> Int {

}

func getRear() -> Int {

}

func isEmpty() -> Bool {

}

func isFull() -> Bool {

}
}

/**
* Your MyCircularDeque object will be instantiated and called as such:
* let obj = MyCircularDeque(k)
* let ret_1: Bool = obj.insertFront(value)
* let ret_2: Bool = obj.insertLast(value)
* let ret_3: Bool = obj.deleteFront()
```

```
* let ret_4: Bool = obj.deleteLast()
* let ret_5: Int = obj.getFront()
* let ret_6: Int = obj.getRear()
* let ret_7: Bool = obj.isEmpty()
* let ret_8: Bool = obj.isFull()
*/
```

Rust Solution:

```
// Problem: Design Circular Deque
// 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 MyCircularDeque {

}



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

fn new(k: i32) -> Self {

}

fn insert_front(&self, value: i32) -> bool {

}

fn insert_last(&self, value: i32) -> bool {

}

fn delete_front(&self) -> bool {
```

```
}

fn delete_last(&self) -> bool {

}

fn get_front(&self) -> i32 {

}

fn get_rear(&self) -> i32 {

}

fn is_empty(&self) -> bool {

}

fn is_full(&self) -> bool {

}
}

/**
* Your MyCircularDeque object will be instantiated and called as such:
* let obj = MyCircularDeque::new(k);
* let ret_1: bool = obj.insert_front(value);
* let ret_2: bool = obj.insert_last(value);
* let ret_3: bool = obj.delete_front();
* let ret_4: bool = obj.delete_last();
* let ret_5: i32 = obj.get_front();
* let ret_6: i32 = obj.get_rear();
* let ret_7: bool = obj.is_empty();
* let ret_8: bool = obj.is_full();
*/
```

Ruby Solution:

```
class MyCircularDeque
```

```
=begin
:type k: Integer
=end
def initialize(k)

end


=begin
:type value: Integer
:rtype: Boolean
=end
def insert_front(value)

end


=begin
:type value: Integer
:rtype: Boolean
=end
def insert_last(value)

end


=begin
:rtype: Boolean
=end
def delete_front()

end


=begin
:rtype: Boolean
=end
def delete_last()

end
```

```
=begin
:rtype: Integer
=end
def get_front()

end


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

end


=begin
:rtype: Boolean
=end
def is_empty()

end


=begin
:rtype: Boolean
=end
def is_full()

end


end

# Your MyCircularDeque object will be instantiated and called as such:
# obj = MyCircularDeque.new(k)
# param_1 = obj.insert_front(value)
# param_2 = obj.insert_last(value)
# param_3 = obj.delete_front()
# param_4 = obj.delete_last()
# param_5 = obj.get_front()
# param_6 = obj.get_rear()
```

```
# param_7 = obj.is_empty()
# param_8 = obj.is_full()
```

PHP Solution:

```
class MyCircularDeque {
/**
* @param Integer $k
*/
function __construct($k) {

}

/**
* @param Integer $value
* @return Boolean
*/
function insertFront($value) {

}

/**
* @param Integer $value
* @return Boolean
*/
function insertLast($value) {

}

/**
* @return Boolean
*/
function deleteFront() {

}

/**
* @return Boolean
*/
function deleteLast() {
```

```
}

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

}

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

}

/**
* @return Boolean
*/
function isEmpty() {

}

/**
* @return Boolean
*/
function isFull() {

}
}

/**
* Your MyCircularDeque object will be instantiated and called as such:
* $obj = MyCircularDeque($k);
* $ret_1 = $obj->insertFront($value);
* $ret_2 = $obj->insertLast($value);
* $ret_3 = $obj->deleteFront();
* $ret_4 = $obj->deleteLast();
* $ret_5 = $obj->getFront();
* $ret_6 = $obj->getRear();
* $ret_7 = $obj->isEmpty();
* $ret_8 = $obj->isFull();
```

```
*/
```

Dart Solution:

```
class MyCircularDeque {

MyCircularDeque(int k) {

}

bool insertFront(int value) {

}

bool insertLast(int value) {

}

bool deleteFront() {

}

bool deleteLast() {

}

int getFront() {

}

int getRear() {

}

bool isEmpty() {

}

bool isFull() {

}
```

```
}

/**
* Your MyCircularDeque object will be instantiated and called as such:
* MyCircularDeque obj = MyCircularDeque(k);
* bool param1 = obj.insertFront(value);
* bool param2 = obj.insertLast(value);
* bool param3 = obj.deleteFront();
* bool param4 = obj.deleteLast();
* int param5 = obj.getFront();
* int param6 = obj.getRear();
* bool param7 = obj.isEmpty();
* bool param8 = obj.isFull();
*/
```

Scala Solution:

```
class MyCircularDeque(_k: Int) {

def insertFront(value: Int): Boolean = {

}

def insertLast(value: Int): Boolean = {

}

def deleteFront(): Boolean = {

}

def deleteLast(): Boolean = {

}

def getFront(): Int = {

}

def getRear(): Int = {
```

```
}

def isEmpty(): Boolean = {

}

def isFull(): Boolean = {

}

}

/**
* Your MyCircularDeque object will be instantiated and called as such:
* val obj = new MyCircularDeque(k)
* val param_1 = obj.insertFront(value)
* val param_2 = obj.insertLast(value)
* val param_3 = obj.deleteFront()
* val param_4 = obj.deleteLast()
* val param_5 = obj.getFront()
* val param_6 = obj.getRear()
* val param_7 = obj.isEmpty()
* val param_8 = obj.isFull()
*/
```

Elixir Solution:

```
defmodule MyCircularDeque do
@spec init_(k :: integer) :: any
def init_(k) do

end

@spec insert_front(value :: integer) :: boolean
def insert_front(value) do

end

@spec insert_last(value :: integer) :: boolean
def insert_last(value) do
```

```
end

@spec delete_front() :: boolean
def delete_front() do

end

@spec delete_last() :: boolean
def delete_last() do

end

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

end

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

end

@spec is_empty() :: boolean
def is_empty() do

end

@spec is_full() :: boolean
def is_full() do

end
end

# Your functions will be called as such:
# MyCircularDeque.init_(k)
# param_1 = MyCircularDeque.insert_front(value)
# param_2 = MyCircularDeque.insert_last(value)
# param_3 = MyCircularDeque.delete_front()
# param_4 = MyCircularDeque.delete_last()
# param_5 = MyCircularDeque.get_front()
# param_6 = MyCircularDeque.get_rear()
# param_7 = MyCircularDeque.is_empty()
```

```
# param_8 = MyCircularDeque.is_full()

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

Erlang Solution:

```
-spec my_circular_deque_init_(K :: integer()) -> any().
my_circular_deque_init_(K) ->
.

-spec my_circular_deque_insert_front(Value :: integer()) -> boolean().
my_circular_deque_insert_front(Value) ->
.

-spec my_circular_deque_insert_last(Value :: integer()) -> boolean().
my_circular_deque_insert_last(Value) ->
.

-spec my_circular_deque_delete_front() -> boolean().
my_circular_deque_delete_front() ->
.

-spec my_circular_deque_delete_last() -> boolean().
my_circular_deque_delete_last() ->
.

-spec my_circular_deque_get_front() -> integer().
my_circular_deque_get_front() ->
.

-spec my_circular_deque_get_rear() -> integer().
my_circular_deque_get_rear() ->
.

-spec my_circular_deque_is_empty() -> boolean().
my_circular_deque_is_empty() ->
.

-spec my_circular_deque_is_full() -> boolean().
my_circular_deque_is_full() ->
```

```
.


%% Your functions will be called as such:
%% my_circular_deque_init_(K),
%% Param_1 = my_circular_deque_insert_front(Value),
%% Param_2 = my_circular_deque_insert_last(Value),
%% Param_3 = my_circular_deque_delete_front(),
%% Param_4 = my_circular_deque_delete_last(),
%% Param_5 = my_circular_deque_get_front(),
%% Param_6 = my_circular_deque_get_rear(),
%% Param_7 = my_circular_deque_is_empty(),
%% Param_8 = my_circular_deque_is_full(),

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

Racket Solution:

```
(define my-circular-deque%
(class object%
(super-new)

; k : exact-integer?
(init-field
k)

; insert-front : exact-integer? -> boolean?
(define/public (insert-front value)
)
; insert-last : exact-integer? -> boolean?
(define/public (insert-last value)
)
; delete-front : -> boolean?
(define/public (delete-front)
)
; delete-last : -> boolean?
(define/public (delete-last)
)
; get-front : -> exact-integer?
(define/public (get-front)
```

```
)
; get-rear : -> exact-integer?
(define/public (get-rear)
)
; is-empty : -> boolean?
(define/public (is-empty)
)
; is-full : -> boolean?
(define/public (is-full)
)))

;; Your my-circular-deque% object will be instantiated and called as such:
;; (define obj (new my-circular-deque% [k k]))
;; (define param_1 (send obj insert-front value))
;; (define param_2 (send obj insert-last value))
;; (define param_3 (send obj delete-front))
;; (define param_4 (send obj delete-last))
;; (define param_5 (send obj get-front))
;; (define param_6 (send obj get-rear))
;; (define param_7 (send obj is-empty))
;; (define param_8 (send obj is-full))
```