Problem 716: Max Stack

Problem Information

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

Problem Description

Design a max stack data structure that supports the stack operations and supports finding the stack's maximum element.

Implement the

MaxStack

class:

MaxStack()

Initializes the stack object.

void push(int x)

Pushes element

x

onto the stack.

int pop()

Removes the element on top of the stack and returns it.

int top()

Gets the element on the top of the stack without removing it.

int peekMax()

Retrieves the maximum element in the stack without removing it.

int popMax()

Retrieves the maximum element in the stack and removes it. If there is more than one maximum element, only remove the

top-most

one.

You must come up with a solution that supports

O(1)

for each

top

call and

O(logn)

for each other call.

Example 1:

Input

["MaxStack", "push", "push", "push", "top", "popMax", "top", "peekMax", "pop", "top"] [[], [5], [1], [5], [], [], [], [], [], []]

Output

[null, null, null, null, 5, 5, 1, 5, 1, 5]

Explanation

MaxStack stk = new MaxStack(); stk.push(5); // [

5

] the top of the stack and the maximum number is 5. stk.push(1); // [

5

,

1

] the top of the stack is 1, but the maximum is 5. stk.push(5); // [5, 1,

5

] the top of the stack is 5, which is also the maximum, because it is the top most one. stk.top(); // return 5, [5, 1,

5

] the stack did not change. stk.popMax(); // return 5, [

5

,

1

] the stack is changed now, and the top is different from the max. stk.top(); // return 1, [

5

,

1

] the stack did not change. stk.peekMax(); // return 5, [

5

,

1

] the stack did not change. stk.pop(); // return 1, [

5

] the top of the stack and the max element is now 5. stk.top(); // return 5, [

5

] the stack did not change.

Constraints:

-10

7

<= x <= 10

7

At most

10

5

calls will be made to

push

,

pop

,

top

,

peekMax

, and

popMax

.

There will be

at least one element

in the stack when

pop

,

top

,

peekMax

, or

popMax

is called.

Code Snippets

C++:

```
class MaxStack {
public:
MaxStack() {

}

void push(int x) {

}

int pop() {

}

int top() {

}

int peekMax() {

}

int popMax() {

}
};

/**
* Your MaxStack object will be instantiated and called as such:
* MaxStack* obj = new MaxStack();
* obj->push(x);
* int param_2 = obj->pop();
* int param_3 = obj->top();
* int param_4 = obj->peekMax();
* int param_5 = obj->popMax();
*/
```

Java:

```
class MaxStack {

public MaxStack() {

}

public void push(int x) {

}

public int pop() {

}

public int top() {

}

public int peekMax() {

}

public int popMax() {

}
}

/**
* Your MaxStack object will be instantiated and called as such:
* MaxStack obj = new MaxStack();
* obj.push(x);
* int param_2 = obj.pop();
* int param_3 = obj.top();
* int param_4 = obj.peekMax();
* int param_5 = obj.popMax();
*/
```

Python3:

```
class MaxStack:
```

```
def __init__(self):


def push(self, x: int) -> None:


def pop(self) -> int:


def top(self) -> int:


def peekMax(self) -> int:


def popMax(self) -> int:



# Your MaxStack object will be instantiated and called as such:
# obj = MaxStack()
# obj.push(x)
# param_2 = obj.pop()
# param_3 = obj.top()
# param_4 = obj.peekMax()
# param_5 = obj.popMax()
```

Python:

```
class MaxStack(object):

def __init__(self):


def push(self, x):
"""
:type x: int
:rtype: None
"""


def pop(self):
```

```
        """
        :rtype: int
        """


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


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


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



# Your MaxStack object will be instantiated and called as such:
# obj = MaxStack()
# obj.push(x)
# param_2 = obj.pop()
# param_3 = obj.top()
# param_4 = obj.peekMax()
# param_5 = obj.popMax()
```

JavaScript:

```
var MaxStack = function() {

};

/**
* @param {number} x
* @return {void}
```

```
*/
MaxStack.prototype.push = function(x) {

};

/**
* @return {number}
*/
MaxStack.prototype.pop = function() {

};

/**
* @return {number}
*/
MaxStack.prototype.top = function() {

};

/**
* @return {number}
*/
MaxStack.prototype.peekMax = function() {

};

/**
* @return {number}
*/
MaxStack.prototype.popMax = function() {

};

/**
* Your MaxStack object will be instantiated and called as such:
* var obj = new MaxStack()
* obj.push(x)
* var param_2 = obj.pop()
* var param_3 = obj.top()
* var param_4 = obj.peekMax()
* var param_5 = obj.popMax()
*/
```

TypeScript:

```
class MaxStack {
constructor() {

}

push(x: number): void {

}

pop(): number {

}

top(): number {

}

peekMax(): number {

}

popMax(): number {

}
}

/**
* Your MaxStack object will be instantiated and called as such:
* var obj = new MaxStack()
* obj.push(x)
* var param_2 = obj.pop()
* var param_3 = obj.top()
* var param_4 = obj.peekMax()
* var param_5 = obj.popMax()
*/
```

C#:

```
public class MaxStack {
```

```
public MaxStack() {

}

public void Push(int x) {

}

public int Pop() {

}

public int Top() {

}

public int PeekMax() {

}

public int PopMax() {

}
}

/**
* Your MaxStack object will be instantiated and called as such:
* MaxStack obj = new MaxStack();
* obj.Push(x);
* int param_2 = obj.Pop();
* int param_3 = obj.Top();
* int param_4 = obj.PeekMax();
* int param_5 = obj.PopMax();
*/
```

C:

```
typedef struct {
```

```
} MaxStack;


MaxStack* maxStackCreate() {

}

void maxStackPush(MaxStack* obj, int x) {

}

int maxStackPop(MaxStack* obj) {

}

int maxStackTop(MaxStack* obj) {

}

int maxStackPeekMax(MaxStack* obj) {

}

int maxStackPopMax(MaxStack* obj) {

}

void maxStackFree(MaxStack* obj) {

}

/**
* Your MaxStack struct will be instantiated and called as such:
* MaxStack* obj = maxStackCreate();
* maxStackPush(obj, x);

* int param_2 = maxStackPop(obj);

* int param_3 = maxStackTop(obj);

* int param_4 = maxStackPeekMax(obj);
```

```
* int param_5 = maxStackPopMax(obj);

* maxStackFree(obj);
*/
```

Go:

```
type MaxStack struct {

}


func Constructor() MaxStack {

}


func (this *MaxStack) Push(x int) {

}


func (this *MaxStack) Pop() int {

}


func (this *MaxStack) Top() int {

}


func (this *MaxStack) PeekMax() int {

}


func (this *MaxStack) PopMax() int {

}
```

```
/**
* Your MaxStack object will be instantiated and called as such:
* obj := Constructor();
* obj.Push(x);
* param_2 := obj.Pop();
* param_3 := obj.Top();
* param_4 := obj.PeekMax();
* param_5 := obj.PopMax();
*/
```

Kotlin:

```
class MaxStack() {

fun push(x: Int) {

}

fun pop(): Int {

}

fun top(): Int {

}

fun peekMax(): Int {

}

fun popMax(): Int {

}

}

/**
* Your MaxStack object will be instantiated and called as such:
* var obj = MaxStack()
* obj.push(x)
* var param_2 = obj.pop()
* var param_3 = obj.top()
```

```
* var param_4 = obj.peekMax()
* var param_5 = obj.popMax()
*/
```

Swift:

```
class MaxStack {

init() {

}

func push(_ x: Int) {

}

func pop() -> Int {

}

func top() -> Int {

}

func peekMax() -> Int {

}

func popMax() -> Int {

}
}

/**
* Your MaxStack object will be instantiated and called as such:
* let obj = MaxStack()
* obj.push(x)
* let ret_2: Int = obj.pop()
* let ret_3: Int = obj.top()
* let ret_4: Int = obj.peekMax()
* let ret_5: Int = obj.popMax()
```

```
*/
```

Rust:

```
struct MaxStack {

}


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

fn new() -> Self {

}

fn push(&self, x: i32) {

}

fn pop(&self) -> i32 {

}

fn top(&self) -> i32 {

}

fn peek_max(&self) -> i32 {

}

fn pop_max(&self) -> i32 {

}
}

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

```
* let obj = MaxStack::new();
* obj.push(x);
* let ret_2: i32 = obj.pop();
* let ret_3: i32 = obj.top();
* let ret_4: i32 = obj.peek_max();
* let ret_5: i32 = obj.pop_max();
*/
```

Ruby:

```
class MaxStack
def initialize()

end


=begin
:type x: Integer
:rtype: Void
=end
def push(x)

end


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

end


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

end


=begin
```

```
:rtype: Integer
=end
def peek_max()

end


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

end



end

# Your MaxStack object will be instantiated and called as such:
# obj = MaxStack.new()
# obj.push(x)
# param_2 = obj.pop()
# param_3 = obj.top()
# param_4 = obj.peek_max()
# param_5 = obj.pop_max()
```

PHP:

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

}

/**
* @param Integer $x
* @return NULL
*/
function push($x) {

}
```

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

}

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

}

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

}

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

}
}

/**
* Your MaxStack object will be instantiated and called as such:
* $obj = MaxStack();
* $obj->push($x);
* $ret_2 = $obj->pop();
* $ret_3 = $obj->top();
* $ret_4 = $obj->peekMax();
* $ret_5 = $obj->popMax();
*/
```

Dart:

```
class MaxStack {

MaxStack() {

}

void push(int x) {

}

int pop() {

}

int top() {

}

int peekMax() {

}

int popMax() {

}
}

/**
* Your MaxStack object will be instantiated and called as such:
* MaxStack obj = MaxStack();
* obj.push(x);
* int param2 = obj.pop();
* int param3 = obj.top();
* int param4 = obj.peekMax();
* int param5 = obj.popMax();
*/
```

Scala:

```
class MaxStack() {

def push(x: Int): Unit = {
```

```
}

def pop(): Int = {

}

def top(): Int = {

}

def peekMax(): Int = {

}

def popMax(): Int = {

}

}

/**
* Your MaxStack object will be instantiated and called as such:
* val obj = new MaxStack()
* obj.push(x)
* val param_2 = obj.pop()
* val param_3 = obj.top()
* val param_4 = obj.peekMax()
* val param_5 = obj.popMax()
*/
```

Elixir:

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

end

@spec push(x :: integer) :: any
def push(x) do

end
```

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

end

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

end

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

end

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

end
end

# Your functions will be called as such:
# MaxStack.init_()
# MaxStack.push(x)
# param_2 = MaxStack.pop()
# param_3 = MaxStack.top()
# param_4 = MaxStack.peek_max()
# param_5 = MaxStack.pop_max()

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

Erlang:

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

-spec max_stack_push(X :: integer()) -> any().
max_stack_push(X) ->
.
```

```
-spec max_stack_pop() -> integer().
max_stack_pop() ->
.

-spec max_stack_top() -> integer().
max_stack_top() ->
.

-spec max_stack_peek_max() -> integer().
max_stack_peek_max() ->
.

-spec max_stack_pop_max() -> integer().
max_stack_pop_max() ->
.


%% Your functions will be called as such:
%% max_stack_init_(),
%% max_stack_push(X),
%% Param_2 = max_stack_pop(),
%% Param_3 = max_stack_top(),
%% Param_4 = max_stack_peek_max(),
%% Param_5 = max_stack_pop_max(),

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

Racket:

```
(define max-stack%
(class object%
(super-new)

(init-field)

; push : exact-integer? -> void?
(define/public (push x)
)
; pop : -> exact-integer?
(define/public (pop)
```

```
)
; top : -> exact-integer?
(define/public (top)
)
; peek-max : -> exact-integer?
(define/public (peek-max)
)
; pop-max : -> exact-integer?
(define/public (pop-max)
)))

;; Your max-stack% object will be instantiated and called as such:
;; (define obj (new max-stack%))
;; (send obj push x)
;; (define param_2 (send obj pop))
;; (define param_3 (send obj top))
;; (define param_4 (send obj peek-max))
;; (define param_5 (send obj pop-max))
```

Solutions

C++ Solution:

```
/*
* Problem: Max Stack
* Difficulty: Hard
* Tags: linked_list, stack
*
* Approach: Optimized algorithm based on problem constraints
* Time Complexity: O(n) to O(n^2) depending on approach
* Space Complexity: O(1) to O(n) depending on approach
*/

class MaxStack {
public:
MaxStack() {

}

void push(int x) {
```

```
}

int pop() {

}

int top() {

}

int peekMax() {

}

int popMax() {

}
};

/**
* Your MaxStack object will be instantiated and called as such:
* MaxStack* obj = new MaxStack();
* obj->push(x);
* int param_2 = obj->pop();
* int param_3 = obj->top();
* int param_4 = obj->peekMax();
* int param_5 = obj->popMax();
*/
```

Java Solution:

```
/**
* Problem: Max Stack
* Difficulty: Hard
* Tags: linked_list, stack
*
* Approach: Optimized algorithm based on problem constraints
* Time Complexity: O(n) to O(n^2) depending on approach
* Space Complexity: O(1) to O(n) depending on approach
*/
```

```
class MaxStack {

public MaxStack() {

}

public void push(int x) {

}

public int pop() {

}

public int top() {

}

public int peekMax() {

}

public int popMax() {

}
}

/**
* Your MaxStack object will be instantiated and called as such:
* MaxStack obj = new MaxStack();
* obj.push(x);
* int param_2 = obj.pop();
* int param_3 = obj.top();
* int param_4 = obj.peekMax();
* int param_5 = obj.popMax();
*/
```

Python3 Solution:

```
"""
Problem: Max Stack
Difficulty: Hard
Tags: linked_list, stack

Approach: Optimized algorithm based on problem constraints
Time Complexity: O(n) to O(n^2) depending on approach
Space Complexity: O(1) to O(n) depending on approach
"""

class MaxStack:

def __init__(self):



def push(self, x: int) -> None:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class MaxStack(object):

def __init__(self):



def push(self, x):
"""
:type x: int
:rtype: None
"""



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



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

```
"""



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



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




# Your MaxStack object will be instantiated and called as such:
# obj = MaxStack()
# obj.push(x)
# param_2 = obj.pop()
# param_3 = obj.top()
# param_4 = obj.peekMax()
# param_5 = obj.popMax()
```

JavaScript Solution:

```
/**
* Problem: Max Stack
* Difficulty: Hard
* Tags: linked_list, stack
*
* Approach: Optimized algorithm based on problem constraints
* Time Complexity: O(n) to O(n^2) depending on approach
* Space Complexity: O(1) to O(n) depending on approach
*/



var MaxStack = function() {

};
```

```
/**
* @param {number} x
* @return {void}
*/
MaxStack.prototype.push = function(x) {

};

/**
* @return {number}
*/
MaxStack.prototype.pop = function() {

};

/**
* @return {number}
*/
MaxStack.prototype.top = function() {

};

/**
* @return {number}
*/
MaxStack.prototype.peekMax = function() {

};

/**
* @return {number}
*/
MaxStack.prototype.popMax = function() {

};

/**
* Your MaxStack object will be instantiated and called as such:
* var obj = new MaxStack()
* obj.push(x)
* var param_2 = obj.pop()
* var param_3 = obj.top()
```

```
* var param_4 = obj.peekMax()
* var param_5 = obj.popMax()
*/
```

TypeScript Solution:

```
/**
* Problem: Max Stack
* Difficulty: Hard
* Tags: linked_list, stack
*
* Approach: Optimized algorithm based on problem constraints
* Time Complexity: O(n) to O(n^2) depending on approach
* Space Complexity: O(1) to O(n) depending on approach
*/

class MaxStack {
constructor() {

}

push(x: number): void {

}

pop(): number {

}

top(): number {

}

peekMax(): number {

}

popMax(): number {

}
}
```

```
/**
* Your MaxStack object will be instantiated and called as such:
* var obj = new MaxStack()
* obj.push(x)
* var param_2 = obj.pop()
* var param_3 = obj.top()
* var param_4 = obj.peekMax()
* var param_5 = obj.popMax()
*/
```

C# Solution:

```
/*
* Problem: Max Stack
* Difficulty: Hard
* Tags: linked_list, stack
*
* Approach: Optimized algorithm based on problem constraints
* Time Complexity: O(n) to O(n^2) depending on approach
* Space Complexity: O(1) to O(n) depending on approach
*/

public class MaxStack {

public MaxStack() {

}

public void Push(int x) {

}

public int Pop() {

}

public int Top() {

}
```

```
public int PeekMax() {

}

public int PopMax() {

}
}

/**
* Your MaxStack object will be instantiated and called as such:
* MaxStack obj = new MaxStack();
* obj.Push(x);
* int param_2 = obj.Pop();
* int param_3 = obj.Top();
* int param_4 = obj.PeekMax();
* int param_5 = obj.PopMax();
*/
```

C Solution:

```
/*
* Problem: Max Stack
* Difficulty: Hard
* Tags: linked_list, stack
*
* Approach: Optimized algorithm based on problem constraints
* Time Complexity: O(n) to O(n^2) depending on approach
* Space Complexity: O(1) to O(n) depending on approach
*/




typedef struct {

} MaxStack;


MaxStack* maxStackCreate() {
```

```
}

void maxStackPush(MaxStack* obj, int x) {

}

int maxStackPop(MaxStack* obj) {

}

int maxStackTop(MaxStack* obj) {

}

int maxStackPeekMax(MaxStack* obj) {

}

int maxStackPopMax(MaxStack* obj) {

}

void maxStackFree(MaxStack* obj) {

}

/**
* Your MaxStack struct will be instantiated and called as such:
* MaxStack* obj = maxStackCreate();
* maxStackPush(obj, x);

* int param_2 = maxStackPop(obj);

* int param_3 = maxStackTop(obj);

* int param_4 = maxStackPeekMax(obj);

* int param_5 = maxStackPopMax(obj);

* maxStackFree(obj);
*/
```

Go Solution:

```
// Problem: Max Stack
// Difficulty: Hard
// Tags: linked_list, stack
//
// Approach: Optimized algorithm based on problem constraints
// Time Complexity: O(n) to O(n^2) depending on approach
// Space Complexity: O(1) to O(n) depending on approach

type MaxStack struct {

}


func Constructor() MaxStack {

}


func (this *MaxStack) Push(x int) {

}


func (this *MaxStack) Pop() int {

}


func (this *MaxStack) Top() int {

}


func (this *MaxStack) PeekMax() int {

}


func (this *MaxStack) PopMax() int {
```

```
}


/**
* Your MaxStack object will be instantiated and called as such:
* obj := Constructor();
* obj.Push(x);
* param_2 := obj.Pop();
* param_3 := obj.Top();
* param_4 := obj.PeekMax();
* param_5 := obj.PopMax();
*/
```

Kotlin Solution:

```
class MaxStack() {

fun push(x: Int) {

}

fun pop(): Int {

}

fun top(): Int {

}

fun peekMax(): Int {

}

fun popMax(): Int {

}

}

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

```
* var obj = MaxStack()
* obj.push(x)
* var param_2 = obj.pop()
* var param_3 = obj.top()
* var param_4 = obj.peekMax()
* var param_5 = obj.popMax()
*/
```

Swift Solution:

```
class MaxStack {

init() {

}

func push(_ x: Int) {

}

func pop() -> Int {

}

func top() -> Int {

}

func peekMax() -> Int {

}

func popMax() -> Int {

}
}

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

```
* obj.push(x)
* let ret_2: Int = obj.pop()
* let ret_3: Int = obj.top()
* let ret_4: Int = obj.peekMax()
* let ret_5: Int = obj.popMax()
*/
```

Rust Solution:

```
// Problem: Max Stack
// Difficulty: Hard
// Tags: linked_list, stack
//
// Approach: Optimized algorithm based on problem constraints
// Time Complexity: O(n) to O(n^2) depending on approach
// Space Complexity: O(1) to O(n) depending on approach

struct MaxStack {

}


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

fn new() -> Self {

}

fn push(&self, x: i32) {

}

fn pop(&self) -> i32 {

}

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

```
}

fn peek_max(&self) -> i32 {

}

fn pop_max(&self) -> i32 {

}
}

/**
* Your MaxStack object will be instantiated and called as such:
* let obj = MaxStack::new();
* obj.push(x);
* let ret_2: i32 = obj.pop();
* let ret_3: i32 = obj.top();
* let ret_4: i32 = obj.peek_max();
* let ret_5: i32 = obj.pop_max();
*/
```

Ruby Solution:

```
class MaxStack
def initialize()

end


=begin
:type x: Integer
:rtype: Void
=end
def push(x)

end


=begin
:rtype: Integer
```

```
=end
def pop()

end


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

end


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

end


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

end


end

# Your MaxStack object will be instantiated and called as such:
# obj = MaxStack.new()
# obj.push(x)
# param_2 = obj.pop()
# param_3 = obj.top()
# param_4 = obj.peek_max()
# param_5 = obj.pop_max()
```

PHP Solution:

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

}

/**
* @param Integer $x
* @return NULL
*/
function push($x) {

}

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

}

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

}

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

}

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

}
}
```

```
/**
* Your MaxStack object will be instantiated and called as such:
* $obj = MaxStack();
* $obj->push($x);
* $ret_2 = $obj->pop();
* $ret_3 = $obj->top();
* $ret_4 = $obj->peekMax();
* $ret_5 = $obj->popMax();
*/
```

Dart Solution:

```
class MaxStack {

MaxStack() {

}

void push(int x) {

}

int pop() {

}

int top() {

}

int peekMax() {

}

int popMax() {

}
}

/**
```

```
* Your MaxStack object will be instantiated and called as such:
* MaxStack obj = MaxStack();
* obj.push(x);
* int param2 = obj.pop();
* int param3 = obj.top();
* int param4 = obj.peekMax();
* int param5 = obj.popMax();
*/
```

Scala Solution:

```
class MaxStack() {

def push(x: Int): Unit = {

}

def pop(): Int = {

}

def top(): Int = {

}

def peekMax(): Int = {

}

def popMax(): Int = {

}

}

/**
* Your MaxStack object will be instantiated and called as such:
* val obj = new MaxStack()
* obj.push(x)
* val param_2 = obj.pop()
* val param_3 = obj.top()
```

```
* val param_4 = obj.peekMax()
* val param_5 = obj.popMax()
*/
```

Elixir Solution:

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

end

@spec push(x :: integer) :: any
def push(x) do

end

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

end

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

end

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

end

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

end
end

# Your functions will be called as such:
# MaxStack.init_()
# MaxStack.push(x)
```

```
# param_2 = MaxStack.pop()
# param_3 = MaxStack.top()
# param_4 = MaxStack.peek_max()
# param_5 = MaxStack.pop_max()

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

Erlang Solution:

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

-spec max_stack_push(X :: integer()) -> any().
max_stack_push(X) ->
.

-spec max_stack_pop() -> integer().
max_stack_pop() ->
.

-spec max_stack_top() -> integer().
max_stack_top() ->
.

-spec max_stack_peek_max() -> integer().
max_stack_peek_max() ->
.

-spec max_stack_pop_max() -> integer().
max_stack_pop_max() ->
.


%% Your functions will be called as such:
%% max_stack_init_(),
%% max_stack_push(X),
%% Param_2 = max_stack_pop(),
%% Param_3 = max_stack_top(),
%% Param_4 = max_stack_peek_max(),
```

```
%% Param_5 = max_stack_pop_max(),

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

Racket Solution:

```
(define max-stack%
(class object%
(super-new)

(init-field)

; push : exact-integer? -> void?
(define/public (push x)
)
; pop : -> exact-integer?
(define/public (pop)
)
; top : -> exact-integer?
(define/public (top)
)
; peek-max : -> exact-integer?
(define/public (peek-max)
)
; pop-max : -> exact-integer?
(define/public (pop-max)
)))

;; Your max-stack% object will be instantiated and called as such:
;; (define obj (new max-stack%))
;; (send obj push x)
;; (define param_2 (send obj pop))
;; (define param_3 (send obj top))
;; (define param_4 (send obj peek-max))
;; (define param_5 (send obj pop-max))
```