Problem 631: Design Excel Sum Formula

Problem Information

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

Problem Description

Design the basic function of

Excel

and implement the function of the sum formula.

Implement the

Excel

class:

Excel(int height, char width)

Initializes the object with the

height

and the

width

of the sheet. The sheet is an integer matrix

mat

of size

height x width

with the row index in the range

[1, height]

and the column index in the range

['A', width]

. All the values should be

zero

initially.

void set(int row, char column, int val)

Changes the value at

mat[row][column]

to be

val

.

int get(int row, char column)

Returns the value at

mat[row][column]

.

int sum(int row, char column, List<String> numbers)

Sets the value at

mat[row][column]

to be the sum of cells represented by

numbers

and returns the value at

mat[row][column]

. This sum formula

should exist

until this cell is overlapped by another value or another sum formula.

numbers[i]

could be on the format:

"ColRow"

that represents a single cell.

For example,

"F7"

represents the cell

mat[7]['F']

.

"ColRow1:ColRow2"

that represents a range of cells. The range will always be a rectangle where

"ColRow1"

represent the position of the top-left cell, and

"ColRow2"

represents the position of the bottom-right cell.

For example,

"B3:F7"

represents the cells

mat[i][j]

for

3 <= i <= 7

and

'B' <= j <= 'F'

.

Note:

You could assume that there will not be any circular sum reference.

For example,

mat[1]['A'] == sum(1, "B")

and

mat[1]['B'] == sum(1, "A")

.

Example 1:

Input

["Excel", "set", "sum", "set", "get"] [[3, "C"], [1, "A", 2], [3, "C", ["A1", "A1:B2"]], [2, "B", 2], [3, "C"]]

Output

[null, null, 4, null, 6]

Explanation

Excel excel = new Excel(3, "C"); // construct a 3*3 2D array with all zero. // A B C // 1 0 0 0 // 2 0 0 0 // 3 0 0 0 excel.set(1, "A", 2); // set mat[1]["A"] to be 2. // A B C // 1 2 0 0 // 2 0 0 0 // 3 0 0 0 excel.sum(3, "C", ["A1", "A1:B2"]); // return 4 // set mat[3]["C"] to be the sum of value at mat[1]["A"] and the values sum of the rectangle range whose top-left cell is mat[1]["A"] and bottom-right cell is mat[2]["B"]. // A B C // 1 2 0 0 // 2 0 0 0 // 3 0 0 4 excel.set(2, "B", 2); // set mat[2]["B"] to be 2. Note mat[3]["C"] should also be changed. // A B C // 1 2 0 0 // 2 0 2 0 // 3 0 0 6 excel.get(3, "C"); // return 6

Constraints:

1 <= height <= 26

'A' <= width <= 'Z'

1 <= row <= height

'A' <= column <= width

-100 <= val <= 100

1 <= numbers.length <= 5

numbers[i]

has the format

"ColRow"

or

"ColRow1:ColRow2"

.

At most

100

calls will be made to

set

,

get

, and

sum

.

Code Snippets

C++:

```
class Excel {
public:
Excel(int height, char width) {

}

void set(int row, char column, int val) {
```

```
}

int get(int row, char column) {

}

int sum(int row, char column, vector<string> numbers) {

}
};

/**
* Your Excel object will be instantiated and called as such:
* Excel* obj = new Excel(height, width);
* obj->set(row,column,val);
* int param_2 = obj->get(row,column);
* int param_3 = obj->sum(row,column,numbers);
*/
```

Java:

```
class Excel {

public Excel(int height, char width) {

}

public void set(int row, char column, int val) {

}

public int get(int row, char column) {

}

public int sum(int row, char column, String[] numbers) {

}
}

/**
```

```
* Your Excel object will be instantiated and called as such:
* Excel obj = new Excel(height, width);
* obj.set(row,column,val);
* int param_2 = obj.get(row,column);
* int param_3 = obj.sum(row,column,numbers);
*/
```

Python3:

```
class Excel:

def __init__(self, height: int, width: str):


def set(self, row: int, column: str, val: int) -> None:


def get(self, row: int, column: str) -> int:


def sum(self, row: int, column: str, numbers: List[str]) -> int:



# Your Excel object will be instantiated and called as such:
# obj = Excel(height, width)
# obj.set(row,column,val)
# param_2 = obj.get(row,column)
# param_3 = obj.sum(row,column,numbers)
```

Python:

```
class Excel(object):

def __init__(self, height, width):
"""
:type height: int
:type width: str
"""


def set(self, row, column, val):
```

```
        """
        :type row: int
        :type column: str
        :type val: int
        :rtype: None
        """


    def get(self, row, column):
        """
        :type row: int
        :type column: str
        :rtype: int
        """


    def sum(self, row, column, numbers):
        """
        :type row: int
        :type column: str
        :type numbers: List[str]
        :rtype: int
        """



# Your Excel object will be instantiated and called as such:
# obj = Excel(height, width)
# obj.set(row,column,val)
# param_2 = obj.get(row,column)
# param_3 = obj.sum(row,column,numbers)
```

JavaScript:

```
/**
* @param {number} height
* @param {character} width
*/
var Excel = function(height, width) {

};
```

```
/**
* @param {number} row
* @param {character} column
* @param {number} val
* @return {void}
*/
Excel.prototype.set = function(row, column, val) {

};

/**
* @param {number} row
* @param {character} column
* @return {number}
*/
Excel.prototype.get = function(row, column) {

};

/**
* @param {number} row
* @param {character} column
* @param {string[]} numbers
* @return {number}
*/
Excel.prototype.sum = function(row, column, numbers) {

};

/**
* Your Excel object will be instantiated and called as such:
* var obj = new Excel(height, width)
* obj.set(row,column,val)
* var param_2 = obj.get(row,column)
* var param_3 = obj.sum(row,column,numbers)
*/
```

TypeScript:

```
class Excel {
constructor(height: number, width: string) {
```

```
}

set(row: number, column: string, val: number): void {

}

get(row: number, column: string): number {

}

sum(row: number, column: string, numbers: string[]): number {

}
}

/**
* Your Excel object will be instantiated and called as such:
* var obj = new Excel(height, width)
* obj.set(row,column,val)
* var param_2 = obj.get(row,column)
* var param_3 = obj.sum(row,column,numbers)
*/
```

C#:

```
public class Excel {

public Excel(int height, char width) {

}

public void Set(int row, char column, int val) {

}

public int Get(int row, char column) {

}

public int Sum(int row, char column, string[] numbers) {

}
```

```
}

/**
* Your Excel object will be instantiated and called as such:
* Excel obj = new Excel(height, width);
* obj.Set(row,column,val);
* int param_2 = obj.Get(row,column);
* int param_3 = obj.Sum(row,column,numbers);
*/
```

C:

```
typedef struct {

} Excel;


Excel* excelCreate(int height, char width) {

}

void excelSet(Excel* obj, int row, char column, int val) {

}

int excelGet(Excel* obj, int row, char column) {

}

int excelSum(Excel* obj, int row, char column, char** numbers, int
numbersSize) {

}

void excelFree(Excel* obj) {

}

/**
```

```
* Your Excel struct will be instantiated and called as such:
* Excel* obj = excelCreate(height, width);
* excelSet(obj, row, column, val);

* int param_2 = excelGet(obj, row, column);

* int param_3 = excelSum(obj, row, column, numbers, numbersSize);

* excelFree(obj);
*/
```

Go:

```
type Excel struct {

}


func Constructor(height int, width byte) Excel {

}


func (this *Excel) Set(row int, column byte, val int) {

}


func (this *Excel) Get(row int, column byte) int {

}


func (this *Excel) Sum(row int, column byte, numbers []string) int {

}


/**
* Your Excel object will be instantiated and called as such:
* obj := Constructor(height, width);
* obj.Set(row,column,val);
```

```
* param_2 := obj.Get(row,column);
* param_3 := obj.Sum(row,column,numbers);
*/
```

Kotlin:

```
class Excel(height: Int, width: Char) {

fun set(row: Int, column: Char, `val`: Int) {

}

fun get(row: Int, column: Char): Int {

}

fun sum(row: Int, column: Char, numbers: Array<String>): Int {

}

}

/**
* Your Excel object will be instantiated and called as such:
* var obj = Excel(height, width)
* obj.set(row,column,`val`)
* var param_2 = obj.get(row,column)
* var param_3 = obj.sum(row,column,numbers)
*/
```

Swift:

```
class Excel {

init(_ height: Int, _ width: Character) {

}

func set(_ row: Int, _ column: Character, _ val: Int) {

}
```

```
func get(_ row: Int, _ column: Character) -> Int {

}

func sum(_ row: Int, _ column: Character, _ numbers: [String]) -> Int {

}
}

/**
* Your Excel object will be instantiated and called as such:
* let obj = Excel(height, width)
* obj.set(row, column, val)
* let ret_2: Int = obj.get(row, column)
* let ret_3: Int = obj.sum(row, column, numbers)
*/
```

Rust:

```
struct Excel {

}


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

fn new(height: i32, width: char) -> Self {

}

fn set(&self, row: i32, column: char, val: i32) {

}

fn get(&self, row: i32, column: char) -> i32 {

}
```

```
fn sum(&self, row: i32, column: char, numbers: Vec<String>) -> i32 {

}
}

/**
* Your Excel object will be instantiated and called as such:
* let obj = Excel::new(height, width);
* obj.set(row, column, val);
* let ret_2: i32 = obj.get(row, column);
* let ret_3: i32 = obj.sum(row, column, numbers);
*/
```

Ruby:

```
class Excel

=begin
:type height: Integer
:type width: Character
=end
def initialize(height, width)

end


=begin
:type row: Integer
:type column: Character
:type val: Integer
:rtype: Void
=end
def set(row, column, val)

end


=begin
:type row: Integer
:type column: Character
:rtype: Integer
```

```
=end
def get(row, column)

end


=begin
:type row: Integer
:type column: Character
:type numbers: String[]
:rtype: Integer
=end
def sum(row, column, numbers)

end


end

# Your Excel object will be instantiated and called as such:
# obj = Excel.new(height, width)
# obj.set(row, column, val)
# param_2 = obj.get(row, column)
# param_3 = obj.sum(row, column, numbers)
```

PHP:

```
class Excel {
/**
* @param Integer $height
* @param String $width
*/
function __construct($height, $width) {

}

/**
* @param Integer $row
* @param String $column
* @param Integer $val
* @return NULL
*/
```

```
function set($row, $column, $val) {

}

/**
* @param Integer $row
* @param String $column
* @return Integer
*/
function get($row, $column) {

}

/**
* @param Integer $row
* @param String $column
* @param String[] $numbers
* @return Integer
*/
function sum($row, $column, $numbers) {

}
}

/**
* Your Excel object will be instantiated and called as such:
* $obj = Excel($height, $width);
* $obj->set($row, $column, $val);
* $ret_2 = $obj->get($row, $column);
* $ret_3 = $obj->sum($row, $column, $numbers);
*/
```

Dart:

```
class Excel {

Excel(int height, String width) {

}

void set(int row, String column, int val) {
```

```
}

int get(int row, String column) {

}

int sum(int row, String column, List<String> numbers) {

}
}

/**
* Your Excel object will be instantiated and called as such:
* Excel obj = Excel(height, width);
* obj.set(row,column,val);
* int param2 = obj.get(row,column);
* int param3 = obj.sum(row,column,numbers);
*/
```

Scala:

```
class Excel(_height: Int, _width: Char) {

def set(row: Int, column: Char, `val`: Int): Unit = {

}

def get(row: Int, column: Char): Int = {

}

def sum(row: Int, column: Char, numbers: Array[String]): Int = {

}

}

/**
* Your Excel object will be instantiated and called as such:
* val obj = new Excel(height, width)
* obj.set(row,column,`val`)
* val param_2 = obj.get(row,column)
```

```
* val param_3 = obj.sum(row,column,numbers)
*/
```

Elixir:

```
defmodule Excel do
@spec init_(height :: integer, width :: char) :: any
def init_(height, width) do

end

@spec set(row :: integer, column :: char, val :: integer) :: any
def set(row, column, val) do

end

@spec get(row :: integer, column :: char) :: integer
def get(row, column) do

end

@spec sum(row :: integer, column :: char, numbers :: [String.t]) :: integer
def sum(row, column, numbers) do

end
end

# Your functions will be called as such:
# Excel.init_(height, width)
# Excel.set(row, column, val)
# param_2 = Excel.get(row, column)
# param_3 = Excel.sum(row, column, numbers)

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

Erlang:

```
-spec excel_init_(Height :: integer(), Width :: char()) -> any().
excel_init_(Height, Width) ->
.
```

```
-spec excel_set(Row :: integer(), Column :: char(), Val :: integer()) ->
any().
excel_set(Row, Column, Val) ->
.

-spec excel_get(Row :: integer(), Column :: char()) -> integer().
excel_get(Row, Column) ->
.

-spec excel_sum(Row :: integer(), Column :: char(), Numbers ::
[unicode:unicode_binary()]) -> integer().
excel_sum(Row, Column, Numbers) ->
.


%% Your functions will be called as such:
%% excel_init_(Height, Width),
%% excel_set(Row, Column, Val),
%% Param_2 = excel_get(Row, Column),
%% Param_3 = excel_sum(Row, Column, Numbers),

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

Racket:

```
(define excel%
(class object%
(super-new)

; height : exact-integer?
; width : char?
(init-field
height
width)

; set : exact-integer? char? exact-integer? -> void?
(define/public (set row column val)
)
; get : exact-integer? char? -> exact-integer?
(define/public (get row column)
)
```

```
; sum : exact-integer? char? (listof string?) -> exact-integer?
(define/public (sum row column numbers)
)))

;; Your excel% object will be instantiated and called as such:
;; (define obj (new excel% [height height] [width width]))
;; (send obj set row column val)
;; (define param_2 (send obj get row column))
;; (define param_3 (send obj sum row column numbers))
```

Solutions

C++ Solution:

```
/*
* Problem: Design Excel Sum Formula
* Difficulty: Hard
* Tags: array, string, graph, hash, sort
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

class Excel {
public:
Excel(int height, char width) {

}

void set(int row, char column, int val) {

}

int get(int row, char column) {

}

int sum(int row, char column, vector<string> numbers) {
```

```
}
};

/**
* Your Excel object will be instantiated and called as such:
* Excel* obj = new Excel(height, width);
* obj->set(row,column,val);
* int param_2 = obj->get(row,column);
* int param_3 = obj->sum(row,column,numbers);
*/
```

Java Solution:

```
/**
* Problem: Design Excel Sum Formula
* Difficulty: Hard
* Tags: array, string, graph, hash, sort
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

class Excel {

public Excel(int height, char width) {

}

public void set(int row, char column, int val) {

}

public int get(int row, char column) {

}

public int sum(int row, char column, String[] numbers) {

}
}
```

```
/**
* Your Excel object will be instantiated and called as such:
* Excel obj = new Excel(height, width);
* obj.set(row,column,val);
* int param_2 = obj.get(row,column);
* int param_3 = obj.sum(row,column,numbers);
*/
```

Python3 Solution:

```
"""
Problem: Design Excel Sum Formula
Difficulty: Hard
Tags: array, string, graph, hash, sort

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(n) for hash map
"""

class Excel:

def __init__(self, height: int, width: str):


def set(self, row: int, column: str, val: int) -> None:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Excel(object):

def __init__(self, height, width):
"""
:type height: int
:type width: str
"""
```

```
def set(self, row, column, val):
"""
:type row: int
:type column: str
:type val: int
:rtype: None
"""


def get(self, row, column):
"""
:type row: int
:type column: str
:rtype: int
"""


def sum(self, row, column, numbers):
"""
:type row: int
:type column: str
:type numbers: List[str]
:rtype: int
"""



# Your Excel object will be instantiated and called as such:
# obj = Excel(height, width)
# obj.set(row,column,val)
# param_2 = obj.get(row,column)
# param_3 = obj.sum(row,column,numbers)
```

JavaScript Solution:

```
/**
* Problem: Design Excel Sum Formula
* Difficulty: Hard
* Tags: array, string, graph, hash, sort
*
* Approach: Use two pointers or sliding window technique
```

```
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

/**
* @param {number} height
* @param {character} width
*/
var Excel = function(height, width) {

};

/**
* @param {number} row
* @param {character} column
* @param {number} val
* @return {void}
*/
Excel.prototype.set = function(row, column, val) {

};

/**
* @param {number} row
* @param {character} column
* @return {number}
*/
Excel.prototype.get = function(row, column) {

};

/**
* @param {number} row
* @param {character} column
* @param {string[]} numbers
* @return {number}
*/
Excel.prototype.sum = function(row, column, numbers) {

};

/**
```

```
* Your Excel object will be instantiated and called as such:
* var obj = new Excel(height, width)
* obj.set(row,column,val)
* var param_2 = obj.get(row,column)
* var param_3 = obj.sum(row,column,numbers)
*/
```

TypeScript Solution:

```
/**
* Problem: Design Excel Sum Formula
* Difficulty: Hard
* Tags: array, string, graph, hash, sort
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

class Excel {
constructor(height: number, width: string) {

}

set(row: number, column: string, val: number): void {

}

get(row: number, column: string): number {

}

sum(row: number, column: string, numbers: string[]): number {

}
}

/**
* Your Excel object will be instantiated and called as such:
* var obj = new Excel(height, width)
* obj.set(row,column,val)
```

```
* var param_2 = obj.get(row,column)
* var param_3 = obj.sum(row,column,numbers)
*/
```

C# Solution:

```
/*
* Problem: Design Excel Sum Formula
* Difficulty: Hard
* Tags: array, string, graph, hash, sort
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

public class Excel {

public Excel(int height, char width) {

}

public void Set(int row, char column, int val) {

}

public int Get(int row, char column) {

}

public int Sum(int row, char column, string[] numbers) {

}
}

/**
* Your Excel object will be instantiated and called as such:
* Excel obj = new Excel(height, width);
* obj.Set(row,column,val);
* int param_2 = obj.Get(row,column);
* int param_3 = obj.Sum(row,column,numbers);
```

```
*/
```

C Solution:

```
/*
* Problem: Design Excel Sum Formula
* Difficulty: Hard
* Tags: array, string, graph, hash, sort
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/




typedef struct {

} Excel;


Excel* excelCreate(int height, char width) {

}

void excelSet(Excel* obj, int row, char column, int val) {

}

int excelGet(Excel* obj, int row, char column) {

}

int excelSum(Excel* obj, int row, char column, char** numbers, int
numbersSize) {

}

void excelFree(Excel* obj) {
```

```
}

/**
* Your Excel struct will be instantiated and called as such:
* Excel* obj = excelCreate(height, width);
* excelSet(obj, row, column, val);

* int param_2 = excelGet(obj, row, column);

* int param_3 = excelSum(obj, row, column, numbers, numbersSize);

* excelFree(obj);
*/
```

Go Solution:

```
// Problem: Design Excel Sum Formula
// Difficulty: Hard
// Tags: array, string, graph, hash, sort
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) for hash map

type Excel struct {

}


func Constructor(height int, width byte) Excel {

}


func (this *Excel) Set(row int, column byte, val int) {

}


func (this *Excel) Get(row int, column byte) int {
```

```
}


func (this *Excel) Sum(row int, column byte, numbers []string) int {

}


/**
* Your Excel object will be instantiated and called as such:
* obj := Constructor(height, width);
* obj.Set(row,column,val);
* param_2 := obj.Get(row,column);
* param_3 := obj.Sum(row,column,numbers);
*/
```

Kotlin Solution:

```
class Excel(height: Int, width: Char) {

fun set(row: Int, column: Char, `val`: Int) {

}

fun get(row: Int, column: Char): Int {

}

fun sum(row: Int, column: Char, numbers: Array<String>): Int {

}

}

/**
* Your Excel object will be instantiated and called as such:
* var obj = Excel(height, width)
* obj.set(row,column,`val`)
* var param_2 = obj.get(row,column)
* var param_3 = obj.sum(row,column,numbers)
*/
```

Swift Solution:

```
class Excel {

init(_ height: Int, _ width: Character) {

}

func set(_ row: Int, _ column: Character, _ val: Int) {

}

func get(_ row: Int, _ column: Character) -> Int {

}

func sum(_ row: Int, _ column: Character, _ numbers: [String]) -> Int {

}
}

/**
* Your Excel object will be instantiated and called as such:
* let obj = Excel(height, width)
* obj.set(row, column, val)
* let ret_2: Int = obj.get(row, column)
* let ret_3: Int = obj.sum(row, column, numbers)
*/
```

Rust Solution:

```
// Problem: Design Excel Sum Formula
// Difficulty: Hard
// Tags: array, string, graph, hash, sort
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) for hash map

struct Excel {
```

```
}


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

fn new(height: i32, width: char) -> Self {

}

fn set(&self, row: i32, column: char, val: i32) {

}

fn get(&self, row: i32, column: char) -> i32 {

}

fn sum(&self, row: i32, column: char, numbers: Vec<String>) -> i32 {

}
}

/**
* Your Excel object will be instantiated and called as such:
* let obj = Excel::new(height, width);
* obj.set(row, column, val);
* let ret_2: i32 = obj.get(row, column);
* let ret_3: i32 = obj.sum(row, column, numbers);
*/
```

Ruby Solution:

```
class Excel

=begin
:type height: Integer
:type width: Character
```

```
=end
def initialize(height, width)

end


=begin
:type row: Integer
:type column: Character
:type val: Integer
:rtype: Void
=end
def set(row, column, val)

end


=begin
:type row: Integer
:type column: Character
:rtype: Integer
=end
def get(row, column)

end


=begin
:type row: Integer
:type column: Character
:type numbers: String[]
:rtype: Integer
=end
def sum(row, column, numbers)

end


end

# Your Excel object will be instantiated and called as such:
# obj = Excel.new(height, width)
```

```
# obj.set(row, column, val)
# param_2 = obj.get(row, column)
# param_3 = obj.sum(row, column, numbers)
```

PHP Solution:

```
class Excel {
/**
* @param Integer $height
* @param String $width
*/
function __construct($height, $width) {

}

/**
* @param Integer $row
* @param String $column
* @param Integer $val
* @return NULL
*/
function set($row, $column, $val) {

}

/**
* @param Integer $row
* @param String $column
* @return Integer
*/
function get($row, $column) {

}

/**
* @param Integer $row
* @param String $column
* @param String[] $numbers
* @return Integer
*/
function sum($row, $column, $numbers) {
```

```
}
}

/**
* Your Excel object will be instantiated and called as such:
* $obj = Excel($height, $width);
* $obj->set($row, $column, $val);
* $ret_2 = $obj->get($row, $column);
* $ret_3 = $obj->sum($row, $column, $numbers);
*/
```

Dart Solution:

```
class Excel {

Excel(int height, String width) {

}

void set(int row, String column, int val) {

}

int get(int row, String column) {

}

int sum(int row, String column, List<String> numbers) {

}
}

/**
* Your Excel object will be instantiated and called as such:
* Excel obj = Excel(height, width);
* obj.set(row,column,val);
* int param2 = obj.get(row,column);
* int param3 = obj.sum(row,column,numbers);
*/
```

Scala Solution:

```
class Excel(_height: Int, _width: Char) {

def set(row: Int, column: Char, `val`: Int): Unit = {

}

def get(row: Int, column: Char): Int = {

}

def sum(row: Int, column: Char, numbers: Array[String]): Int = {

}

}

/**
* Your Excel object will be instantiated and called as such:
* val obj = new Excel(height, width)
* obj.set(row,column,`val`)
* val param_2 = obj.get(row,column)
* val param_3 = obj.sum(row,column,numbers)
*/
```

Elixir Solution:

```
defmodule Excel do
@spec init_(height :: integer, width :: char) :: any
def init_(height, width) do

end

@spec set(row :: integer, column :: char, val :: integer) :: any
def set(row, column, val) do

end

@spec get(row :: integer, column :: char) :: integer
def get(row, column) do

end
```

```
@spec sum(row :: integer, column :: char, numbers :: [String.t]) :: integer
def sum(row, column, numbers) do

end
end

# Your functions will be called as such:
# Excel.init_(height, width)
# Excel.set(row, column, val)
# param_2 = Excel.get(row, column)
# param_3 = Excel.sum(row, column, numbers)

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

Erlang Solution:

```
-spec excel_init_(Height :: integer(), Width :: char()) -> any().
excel_init_(Height, Width) ->
.

-spec excel_set(Row :: integer(), Column :: char(), Val :: integer()) ->
any().
excel_set(Row, Column, Val) ->
.

-spec excel_get(Row :: integer(), Column :: char()) -> integer().
excel_get(Row, Column) ->
.

-spec excel_sum(Row :: integer(), Column :: char(), Numbers ::
[unicode:unicode_binary()]) -> integer().
excel_sum(Row, Column, Numbers) ->
.


%% Your functions will be called as such:
%% excel_init_(Height, Width),
%% excel_set(Row, Column, Val),
%% Param_2 = excel_get(Row, Column),
```

```
%% Param_3 = excel_sum(Row, Column, Numbers),

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

Racket Solution:

```
(define excel%
(class object%
(super-new)

; height : exact-integer?
; width : char?
(init-field
height
width)

; set : exact-integer? char? exact-integer? -> void?
(define/public (set row column val)
)
; get : exact-integer? char? -> exact-integer?
(define/public (get row column)
)
; sum : exact-integer? char? (listof string?) -> exact-integer?
(define/public (sum row column numbers)
)))

;; Your excel% object will be instantiated and called as such:
;; (define obj (new excel% [height height] [width width]))
;; (send obj set row column val)
;; (define param_2 (send obj get row column))
;; (define param_3 (send obj sum row column numbers))
```