Problem 3484: Design Spreadsheet

Problem Information

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

Problem Description

A spreadsheet is a grid with 26 columns (labeled from

'A'

to

'Z'

) and a given number of

rows

. Each cell in the spreadsheet can hold an integer value between 0 and 10

5

.

Implement the

Spreadsheet

class:

Spreadsheet(int rows)

Initializes a spreadsheet with 26 columns (labeled

'A'

to

'Z'

) and the specified number of rows. All cells are initially set to 0.

void setCell(String cell, int value)

Sets the value of the specified

cell

. The cell reference is provided in the format

"AX"

(e.g.,

"A1"

,

"B10"

), where the letter represents the column (from

'A'

to

'Z'

) and the number represents a

1-indexed

row.

void resetCell(String cell)

Resets the specified cell to 0.

int getValue(String formula)

Evaluates a formula of the form

"=X+Y"

, where

X

and

Y

are

either

cell references or non-negative integers, and returns the computed sum.

Note:

If

getValue

references a cell that has not been explicitly set using

setCell

, its value is considered 0.

Example 1:

Input:

["Spreadsheet", "getValue", "setCell", "getValue", "setCell", "getValue", "resetCell", "getValue"]

[[3], ["=5+7"], ["A1", 10], ["=A1+6"], ["B2", 15], ["=A1+B2"], ["A1"], ["=A1+B2"]]

Output:

[null, 12, null, 16, null, 25, null, 15]

Explanation

Spreadsheet spreadsheet = new Spreadsheet(3); // Initializes a spreadsheet with 3 rows and 26 columns

spreadsheet.getValue("=5+7"); // returns 12 (5+7)

spreadsheet.setCell("A1", 10); // sets A1 to 10

spreadsheet.getValue("=A1+6"); // returns 16 (10+6)

spreadsheet.setCell("B2", 15); // sets B2 to 15

spreadsheet.getValue("=A1+B2"); // returns 25 (10+15)

spreadsheet.resetCell("A1"); // resets A1 to 0

spreadsheet.getValue("=A1+B2"); // returns 15 (0+15)

Constraints:

1 <= rows <= 10

3

0 <= value <= 10

5

The formula is always in the format

"=X+Y"

, where

X

and

Y

are either valid cell references or

non-negative

integers with values less than or equal to

10

5

.

Each cell reference consists of a capital letter from

'A'

to

'Z'

followed by a row number between

1

and

rows

.

At most

10

4

calls will be made in

total

to

setCell

,

resetCell

, and

getValue

.

Code Snippets

C++:

```
class Spreadsheet {
public:
Spreadsheet(int rows) {

}

void setCell(string cell, int value) {

}

void resetCell(string cell) {

}

int getValue(string formula) {

}
};

/**
* Your Spreadsheet object will be instantiated and called as such:
* Spreadsheet* obj = new Spreadsheet(rows);
* obj->setCell(cell,value);
* obj->resetCell(cell);
* int param_3 = obj->getValue(formula);
*/
```

Java:

```
class Spreadsheet {

public Spreadsheet(int rows) {

}

public void setCell(String cell, int value) {

}

public void resetCell(String cell) {

}

public int getValue(String formula) {

}
}

/**
* Your Spreadsheet object will be instantiated and called as such:
* Spreadsheet obj = new Spreadsheet(rows);
* obj.setCell(cell,value);
* obj.resetCell(cell);
* int param_3 = obj.getValue(formula);
*/
```

Python3:

```
class Spreadsheet:

def __init__(self, rows: int):



def setCell(self, cell: str, value: int) -> None:



def resetCell(self, cell: str) -> None:



def getValue(self, formula: str) -> int:




# Your Spreadsheet object will be instantiated and called as such:
# obj = Spreadsheet(rows)
# obj.setCell(cell,value)
# obj.resetCell(cell)
# param_3 = obj.getValue(formula)
```

Python:

```
class Spreadsheet(object):

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

```
def setCell(self, cell, value):
"""
:type cell: str
:type value: int
:rtype: None
"""



def resetCell(self, cell):
"""
:type cell: str
:rtype: None
"""



def getValue(self, formula):
"""
:type formula: str
:rtype: int
"""




# Your Spreadsheet object will be instantiated and called as such:
# obj = Spreadsheet(rows)
# obj.setCell(cell,value)
# obj.resetCell(cell)
# param_3 = obj.getValue(formula)
```

JavaScript:

```
/**
* @param {number} rows
*/
var Spreadsheet = function(rows) {

};

/**
* @param {string} cell
* @param {number} value
* @return {void}
*/
Spreadsheet.prototype.setCell = function(cell, value) {

};

/**
* @param {string} cell
* @return {void}
*/
Spreadsheet.prototype.resetCell = function(cell) {

};

/**
* @param {string} formula
* @return {number}
*/
Spreadsheet.prototype.getValue = function(formula) {

};

/**
* Your Spreadsheet object will be instantiated and called as such:
* var obj = new Spreadsheet(rows)
* obj.setCell(cell,value)
* obj.resetCell(cell)
* var param_3 = obj.getValue(formula)
*/
```

TypeScript:

```
class Spreadsheet {
constructor(rows: number) {

}

setCell(cell: string, value: number): void {

}

resetCell(cell: string): void {
}

getValue(formula: string): number {

}
}

/**
* Your Spreadsheet object will be instantiated and called as such:
* var obj = new Spreadsheet(rows)
* obj.setCell(cell,value)
* obj.resetCell(cell)
* var param_3 = obj.getValue(formula)
*/
```

C#:

```
public class Spreadsheet {

public Spreadsheet(int rows) {

}

public void SetCell(string cell, int value) {

}

public void ResetCell(string cell) {

}

public int GetValue(string formula) {

}
}

/**
* Your Spreadsheet object will be instantiated and called as such:
* Spreadsheet obj = new Spreadsheet(rows);
* obj.SetCell(cell,value);
* obj.ResetCell(cell);
* int param_3 = obj.GetValue(formula);
*/
```

C:

```
typedef struct {

} Spreadsheet;


Spreadsheet* spreadsheetCreate(int rows) {

}

void spreadsheetSetCell(Spreadsheet* obj, char* cell, int value) {

}

void spreadsheetResetCell(Spreadsheet* obj, char* cell) {

}

int spreadsheetGetValue(Spreadsheet* obj, char* formula) {

}

void spreadsheetFree(Spreadsheet* obj) {

}

/**
* Your Spreadsheet struct will be instantiated and called as such:
* Spreadsheet* obj = spreadsheetCreate(rows);
* spreadsheetSetCell(obj, cell, value);

* spreadsheetResetCell(obj, cell);

* int param_3 = spreadsheetGetValue(obj, formula);
* spreadsheetFree(obj);
*/
```

Go:

```
type Spreadsheet struct {

}


func Constructor(rows int) Spreadsheet {

}


func (this *Spreadsheet) SetCell(cell string, value int) {

}


func (this *Spreadsheet) ResetCell(cell string) {

}


func (this *Spreadsheet) GetValue(formula string) int {

}


/**
* Your Spreadsheet object will be instantiated and called as such:
* obj := Constructor(rows);
* obj.SetCell(cell,value);
* obj.ResetCell(cell);
* param_3 := obj.GetValue(formula);
*/
```

Kotlin:

```
class Spreadsheet(rows: Int) {
```

```
fun setCell(cell: String, value: Int) {

}

fun resetCell(cell: String) {

}

fun getValue(formula: String): Int {

}

}

/**
* Your Spreadsheet object will be instantiated and called as such:
* var obj = Spreadsheet(rows)
* obj.setCell(cell,value)
* obj.resetCell(cell)
* var param_3 = obj.getValue(formula)
*/
```

Swift:

```
class Spreadsheet {

init(_ rows: Int) {

}

func setCell(_ cell: String, _ value: Int) {

}

func resetCell(_ cell: String) {

}

func getValue(_ formula: String) -> Int {

}
}

/**
* Your Spreadsheet object will be instantiated and called as such:
* let obj = Spreadsheet(rows)
* obj.setCell(cell, value)
* obj.resetCell(cell)
* let ret_3: Int = obj.getValue(formula)
*/
```

Rust:

```
struct Spreadsheet {

}



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

fn new(rows: i32) -> Self {

}

fn set_cell(&self, cell: String, value: i32) {

}

fn reset_cell(&self, cell: String) {

}

fn get_value(&self, formula: String) -> i32 {

}
}

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

```
* let obj = Spreadsheet::new(rows);
* obj.set_cell(cell, value);
* obj.reset_cell(cell);
* let ret_3: i32 = obj.get_value(formula);
*/
```

Ruby:

```
class Spreadsheet

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

end



=begin
:type cell: String
:type value: Integer
:rtype: Void
=end
def set_cell(cell, value)

end



=begin
:type cell: String
:rtype: Void
=end
def reset_cell(cell)

end



=begin
:type formula: String
:rtype: Integer
=end
def get_value(formula)
```

```


end


end

# Your Spreadsheet object will be instantiated and called as such:
# obj = Spreadsheet.new(rows)
# obj.set_cell(cell, value)
# obj.reset_cell(cell)
# param_3 = obj.get_value(formula)
```

PHP:

```
class Spreadsheet {
/**
* @param Integer $rows
*/
function __construct($rows) {

}

/**
* @param String $cell
* @param Integer $value
* @return NULL
*/
function setCell($cell, $value) {

}

/**
* @param String $cell
* @return NULL
*/
function resetCell($cell) {

}

/**
* @param String $formula
* @return Integer
*/
function getValue($formula) {

}
}

/**
* Your Spreadsheet object will be instantiated and called as such:
* $obj = Spreadsheet($rows);
* $obj->setCell($cell, $value);
* $obj->resetCell($cell);
* $ret_3 = $obj->getValue($formula);
*/
```

Dart:

```
class Spreadsheet {

Spreadsheet(int rows) {

}

void setCell(String cell, int value) {

}

void resetCell(String cell) {

}

int getValue(String formula) {

}
}

/**
* Your Spreadsheet object will be instantiated and called as such:
* Spreadsheet obj = Spreadsheet(rows);
* obj.setCell(cell,value);
* obj.resetCell(cell);
* int param3 = obj.getValue(formula);
*/
```

Scala:

```
class Spreadsheet(_rows: Int) {

def setCell(cell: String, value: Int): Unit = {

}

def resetCell(cell: String): Unit = {

}

def getValue(formula: String): Int = {

}

}

/**
* Your Spreadsheet object will be instantiated and called as such:
* val obj = new Spreadsheet(rows)
* obj.setCell(cell,value)
* obj.resetCell(cell)
* val param_3 = obj.getValue(formula)
*/
```

Elixir:

```
defmodule Spreadsheet do
@spec init_(rows :: integer) :: any
def init_(rows) do

end

@spec set_cell(cell :: String.t, value :: integer) :: any
def set_cell(cell, value) do

end

@spec reset_cell(cell :: String.t) :: any
def reset_cell(cell) do
```

```
end

@spec get_value(formula :: String.t) :: integer
def get_value(formula) do

end
end

# Your functions will be called as such:
# Spreadsheet.init_(rows)
# Spreadsheet.set_cell(cell, value)
# Spreadsheet.reset_cell(cell)
# param_3 = Spreadsheet.get_value(formula)

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

Erlang:

```
-spec spreadsheet_init_(Rows :: integer()) -> any().
spreadsheet_init_(Rows) ->
.

-spec spreadsheet_set_cell(Cell :: unicode:unicode_binary(), Value ::
integer()) -> any().
spreadsheet_set_cell(Cell, Value) ->
.

-spec spreadsheet_reset_cell(Cell :: unicode:unicode_binary()) -> any().
spreadsheet_reset_cell(Cell) ->
.

-spec spreadsheet_get_value(Formula :: unicode:unicode_binary()) ->
integer().
spreadsheet_get_value(Formula) ->
.


%% Your functions will be called as such:
%% spreadsheet_init_(Rows),
%% spreadsheet_set_cell(Cell, Value),
%% spreadsheet_reset_cell(Cell),
%% Param_3 = spreadsheet_get_value(Formula),

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

Racket:

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

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

; set-cell : string? exact-integer? -> void?
(define/public (set-cell cell value)
)
; reset-cell : string? -> void?
(define/public (reset-cell cell)
)
; get-value : string? -> exact-integer?
(define/public (get-value formula)
)))

;; Your spreadsheet% object will be instantiated and called as such:
;; (define obj (new spreadsheet% [rows rows]))
;; (send obj set-cell cell value)
;; (send obj reset-cell cell)
;; (define param_3 (send obj get-value formula))
```

Solutions

C++ Solution:

```
/*
* Problem: Design Spreadsheet
* Difficulty: Medium
* Tags: array, string, hash
*
```

```
* 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 Spreadsheet {
public:
Spreadsheet(int rows) {

}

void setCell(string cell, int value) {

}

void resetCell(string cell) {

}

int getValue(string formula) {

}
};

/**
* Your Spreadsheet object will be instantiated and called as such:
* Spreadsheet* obj = new Spreadsheet(rows);
* obj->setCell(cell,value);
* obj->resetCell(cell);
* int param_3 = obj->getValue(formula);
*/
```

Java Solution:

```
/**
* Problem: Design Spreadsheet
* Difficulty: Medium
* Tags: array, string, hash
*
* 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 Spreadsheet {

public Spreadsheet(int rows) {

}

public void setCell(String cell, int value) {

}

public void resetCell(String cell) {

}

public int getValue(String formula) {

}
}

/**
* Your Spreadsheet object will be instantiated and called as such:
* Spreadsheet obj = new Spreadsheet(rows);
* obj.setCell(cell,value);
* obj.resetCell(cell);
* int param_3 = obj.getValue(formula);
*/
```

Python3 Solution:

```
"""
Problem: Design Spreadsheet
Difficulty: Medium
Tags: array, string, hash

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 Spreadsheet:

def __init__(self, rows: int):


def setCell(self, cell: str, value: int) -> None:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Spreadsheet(object):

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


def setCell(self, cell, value):
"""
:type cell: str
:type value: int
:rtype: None
"""


def resetCell(self, cell):
"""
:type cell: str
:rtype: None
"""


def getValue(self, formula):
"""
:type formula: str
:rtype: int
"""
```

```
# Your Spreadsheet object will be instantiated and called as such:
# obj = Spreadsheet(rows)
# obj.setCell(cell,value)
# obj.resetCell(cell)
# param_3 = obj.getValue(formula)
```

JavaScript Solution:

```
/**
* Problem: Design Spreadsheet
* Difficulty: Medium
* Tags: array, string, hash
*
* 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} rows
*/
var Spreadsheet = function(rows) {

};

/**
* @param {string} cell
* @param {number} value
* @return {void}
*/
Spreadsheet.prototype.setCell = function(cell, value) {

};

/**
* @param {string} cell
* @return {void}
*/
Spreadsheet.prototype.resetCell = function(cell) {
};

/**
* @param {string} formula
* @return {number}
*/
Spreadsheet.prototype.getValue = function(formula) {

};

/**
* Your Spreadsheet object will be instantiated and called as such:
* var obj = new Spreadsheet(rows)
* obj.setCell(cell,value)
* obj.resetCell(cell)
* var param_3 = obj.getValue(formula)
*/
```

TypeScript Solution:

```
/**
* Problem: Design Spreadsheet
* Difficulty: Medium
* Tags: array, string, hash
*
* 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 Spreadsheet {
constructor(rows: number) {

}

setCell(cell: string, value: number): void {

}

resetCell(cell: string): void {
}

getValue(formula: string): number {

}
}

/**
* Your Spreadsheet object will be instantiated and called as such:
* var obj = new Spreadsheet(rows)
* obj.setCell(cell,value)
* obj.resetCell(cell)
* var param_3 = obj.getValue(formula)
*/
```

C# Solution:

```
/*
* Problem: Design Spreadsheet
* Difficulty: Medium
* Tags: array, string, hash
*
* 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 Spreadsheet {

public Spreadsheet(int rows) {

}

public void SetCell(string cell, int value) {

}

public void ResetCell(string cell) {

}
public int GetValue(string formula) {

}
}

/**
* Your Spreadsheet object will be instantiated and called as such:
* Spreadsheet obj = new Spreadsheet(rows);
* obj.SetCell(cell,value);
* obj.ResetCell(cell);
* int param_3 = obj.GetValue(formula);
*/
```

C Solution:

```
/*
* Problem: Design Spreadsheet
* Difficulty: Medium
* Tags: array, string, hash
*
* 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 {

} Spreadsheet;


Spreadsheet* spreadsheetCreate(int rows) {

}

void spreadsheetSetCell(Spreadsheet* obj, char* cell, int value) {

}
```

}

```
void spreadsheetResetCell(Spreadsheet* obj, char* cell) {

}

int spreadsheetGetValue(Spreadsheet* obj, char* formula) {

}

void spreadsheetFree(Spreadsheet* obj) {

}

/**
* Your Spreadsheet struct will be instantiated and called as such:
* Spreadsheet* obj = spreadsheetCreate(rows);
* spreadsheetSetCell(obj, cell, value);

* spreadsheetResetCell(obj, cell);

* int param_3 = spreadsheetGetValue(obj, formula);

* spreadsheetFree(obj);
*/
```

Go Solution:

```
// Problem: Design Spreadsheet
// Difficulty: Medium
// Tags: array, string, hash
//
// 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 Spreadsheet struct {

}



func Constructor(rows int) Spreadsheet {
}


func (this *Spreadsheet) SetCell(cell string, value int) {

}


func (this *Spreadsheet) ResetCell(cell string) {

}


func (this *Spreadsheet) GetValue(formula string) int {

}


/**
* Your Spreadsheet object will be instantiated and called as such:
* obj := Constructor(rows);
* obj.SetCell(cell,value);
* obj.ResetCell(cell);
* param_3 := obj.GetValue(formula);
*/
```

Kotlin Solution:

```
class Spreadsheet(rows: Int) {

fun setCell(cell: String, value: Int) {

}

fun resetCell(cell: String) {

}

fun getValue(formula: String): Int {

}

}

/**
* Your Spreadsheet object will be instantiated and called as such:
* var obj = Spreadsheet(rows)
* obj.setCell(cell,value)
* obj.resetCell(cell)
* var param_3 = obj.getValue(formula)
*/
```

Swift Solution:

```

class Spreadsheet {

init(_ rows: Int) {

}

func setCell(_ cell: String, _ value: Int) {

}

func resetCell(_ cell: String) {

}

func getValue(_ formula: String) -> Int {

}
}

/**
* Your Spreadsheet object will be instantiated and called as such:
* let obj = Spreadsheet(rows)
* obj.setCell(cell, value)
* obj.resetCell(cell)
* let ret_3: Int = obj.getValue(formula)
*/
```

Rust Solution:

```
// Problem: Design Spreadsheet
// Difficulty: Medium
// Tags: array, string, hash
//
// 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 Spreadsheet {

}



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

fn new(rows: i32) -> Self {

}

fn set_cell(&self, cell: String, value: i32) {

}

fn reset_cell(&self, cell: String) {

}

fn get_value(&self, formula: String) -> i32 {

}
}

/**
* Your Spreadsheet object will be instantiated and called as such:
* let obj = Spreadsheet::new(rows);
* obj.set_cell(cell, value);
* obj.reset_cell(cell);
* let ret_3: i32 = obj.get_value(formula);
*/
```

Ruby Solution:

```
class Spreadsheet

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

end



=begin
:type cell: String
:type value: Integer
:rtype: Void
=end
def set_cell(cell, value)

end



=begin
:type cell: String
:rtype: Void
=end
def reset_cell(cell)

end



=begin
:type formula: String
:rtype: Integer
=end
def get_value(formula)
```

```
end


end

# Your Spreadsheet object will be instantiated and called as such:
# obj = Spreadsheet.new(rows)
# obj.set_cell(cell, value)
# obj.reset_cell(cell)
# param_3 = obj.get_value(formula)
```

PHP Solution:

```
class Spreadsheet {
/**
* @param Integer $rows
*/
function __construct($rows) {

}

/**
* @param String $cell
* @param Integer $value
* @return NULL
*/
function setCell($cell, $value) {

}

/**
* @param String $cell
* @return NULL
*/
function resetCell($cell) {

}

/**
* @param String $formula
* @return Integer
*/
function getValue($formula) {

}
}

/**
* Your Spreadsheet object will be instantiated and called as such:
* $obj = Spreadsheet($rows);
* $obj->setCell($cell, $value);
* $obj->resetCell($cell);
* $ret_3 = $obj->getValue($formula);
*/
```

Dart Solution:

```
class Spreadsheet {

Spreadsheet(int rows) {

}

void setCell(String cell, int value) {

}

void resetCell(String cell) {

}

int getValue(String formula) {

}
}

/**
* Your Spreadsheet object will be instantiated and called as such:
* Spreadsheet obj = Spreadsheet(rows);
* obj.setCell(cell,value);
* obj.resetCell(cell);
* int param3 = obj.getValue(formula);
```

```
*/
```

Scala Solution:

```
class Spreadsheet(_rows: Int) {

def setCell(cell: String, value: Int): Unit = {

}

def resetCell(cell: String): Unit = {

}

def getValue(formula: String): Int = {

}

}

/**
* Your Spreadsheet object will be instantiated and called as such:
* val obj = new Spreadsheet(rows)
* obj.setCell(cell,value)
* obj.resetCell(cell)
* val param_3 = obj.getValue(formula)
*/
```

Elixir Solution:

```
defmodule Spreadsheet do
@spec init_(rows :: integer) :: any
def init_(rows) do

end

@spec set_cell(cell :: String.t, value :: integer) :: any
def set_cell(cell, value) do

end
```

```
@spec reset_cell(cell :: String.t) :: any
def reset_cell(cell) do

end

@spec get_value(formula :: String.t) :: integer
def get_value(formula) do

end
end

# Your functions will be called as such:
# Spreadsheet.init_(rows)
# Spreadsheet.set_cell(cell, value)
# Spreadsheet.reset_cell(cell)
# param_3 = Spreadsheet.get_value(formula)

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

Erlang Solution:

```
-spec spreadsheet_init_(Rows :: integer()) -> any().
spreadsheet_init_(Rows) ->
.

-spec spreadsheet_set_cell(Cell :: unicode:unicode_binary(), Value ::
integer()) -> any().
spreadsheet_set_cell(Cell, Value) ->
.

-spec spreadsheet_reset_cell(Cell :: unicode:unicode_binary()) -> any().
spreadsheet_reset_cell(Cell) ->
.

-spec spreadsheet_get_value(Formula :: unicode:unicode_binary()) ->
integer().
spreadsheet_get_value(Formula) ->
.
```

```
%% Your functions will be called as such:
%% spreadsheet_init_(Rows),
%% spreadsheet_set_cell(Cell, Value),
%% spreadsheet_reset_cell(Cell),
%% Param_3 = spreadsheet_get_value(Formula),

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

Racket Solution:

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

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

; set-cell : string? exact-integer? -> void?
(define/public (set-cell cell value)
)
; reset-cell : string? -> void?
(define/public (reset-cell cell)
)
; get-value : string? -> exact-integer?
(define/public (get-value formula)
)))

;; Your spreadsheet% object will be instantiated and called as such:
;; (define obj (new spreadsheet% [rows rows]))
;; (send obj set-cell cell value)
;; (send obj reset-cell cell)
;; (define param_3 (send obj get-value formula))
```