Problem 2296: Design a Text Editor

Problem Information

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

Problem Description

Design a text editor with a cursor that can do the following:

Add

text to where the cursor is.

Delete

text from where the cursor is (simulating the backspace key).

Move

the cursor either left or right.

When deleting text, only characters to the left of the cursor will be deleted. The cursor will also remain within the actual text and cannot be moved beyond it. More formally, we have that

0 <= cursor.position <= currentText.length

always holds.

Implement the

TextEditor

class:

TextEditor()

Initializes the object with empty text.

void addText(string text)

Appends

text

to where the cursor is. The cursor ends to the right of

text

.

int deleteText(int k)

Deletes

k

characters to the left of the cursor. Returns the number of characters actually deleted.

string cursorLeft(int k)

Moves the cursor to the left

k

times. Returns the last

min(10, len)

characters to the left of the cursor, where

len

is the number of characters to the left of the cursor.

string cursorRight(int k)

Moves the cursor to the right

k

times. Returns the last

min(10, len)

characters to the left of the cursor, where

len

is the number of characters to the left of the cursor.

Example 1:

Input

["TextEditor", "addText", "deleteText", "addText", "cursorRight", "cursorLeft", "deleteText", "cursorLeft", "cursorRight"] [[], ["leetcode"], [4], ["practice"], [3], [8], [10], [2], [6]]

Output

[null, null, 4, null, "etpractice", "leet", 4, "", "practi"]

Explanation

TextEditor textEditor = new TextEditor(); // The current text is "|". (The '|' character represents the cursor) textEditor.addText("leetcode"); // The current text is "leetcode|". textEditor.deleteText(4); // return 4 // The current text is "leet|". // 4 characters were deleted. textEditor.addText("practice"); // The current text is "leetpractice|". textEditor.cursorRight(3); // return "etpractice" // The current text is "leetpractice|". // The cursor cannot be moved beyond the actual text and thus did not move. // "etpractice" is the last 10 characters to the left of the cursor. textEditor.cursorLeft(8); // return "leet" // The current text is "leet|practice". // "leet" is the last min(10, 4) = 4 characters to the left of the cursor. textEditor.deleteText(10); // return 4 // The current text is "|practice". // Only 4 characters were deleted. textEditor.cursorLeft(2); //

return "" // The current text is "|practice". // The cursor cannot be moved beyond the actual text and thus did not move. // "" is the last min(10, 0) = 0 characters to the left of the cursor. textEditor.cursorRight(6); // return "practi" // The current text is "practi|ce". // "practi" is the last min(10, 6) = 6 characters to the left of the cursor.

Constraints:

1 <= text.length, k <= 40

text

consists of lowercase English letters.

At most

2 * 10

4

calls

in total

will be made to

addText

,

deleteText

,

cursorLeft

and

cursorRight

.

Follow-up:

Could you find a solution with time complexity of

O(k)

per call?

Code Snippets

C++:

```
class TextEditor {
public:
TextEditor() {

}

void addText(string text) {

}

int deleteText(int k) {

}

string cursorLeft(int k) {

}

string cursorRight(int k) {

}
};

/**
* Your TextEditor object will be instantiated and called as such:
* TextEditor* obj = new TextEditor();
```

```
* obj->addText(text);
* int param_2 = obj->deleteText(k);
* string param_3 = obj->cursorLeft(k);
* string param_4 = obj->cursorRight(k);
*/
```

Java:

```
class TextEditor {

public TextEditor() {

}

public void addText(String text) {

}

public int deleteText(int k) {

}

public String cursorLeft(int k) {

}

public String cursorRight(int k) {

}
}

/**
* Your TextEditor object will be instantiated and called as such:
* TextEditor obj = new TextEditor();
* obj.addText(text);
* int param_2 = obj.deleteText(k);
* String param_3 = obj.cursorLeft(k);
* String param_4 = obj.cursorRight(k);
*/
```

Python3:

```
class TextEditor:

def __init__(self):


def addText(self, text: str) -> None:


def deleteText(self, k: int) -> int:


def cursorLeft(self, k: int) -> str:


def cursorRight(self, k: int) -> str:



# Your TextEditor object will be instantiated and called as such:
# obj = TextEditor()
# obj.addText(text)
# param_2 = obj.deleteText(k)
# param_3 = obj.cursorLeft(k)
# param_4 = obj.cursorRight(k)
```

Python:

```
class TextEditor(object):

def __init__(self):


def addText(self, text):
"""
:type text: str
:rtype: None
"""


def deleteText(self, k):
"""
:type k: int
:rtype: int
```

```
"""


def cursorLeft(self, k):
"""
:type k: int
:rtype: str
"""


def cursorRight(self, k):
"""
:type k: int
:rtype: str
"""



# Your TextEditor object will be instantiated and called as such:
# obj = TextEditor()
# obj.addText(text)
# param_2 = obj.deleteText(k)
# param_3 = obj.cursorLeft(k)
# param_4 = obj.cursorRight(k)
```

JavaScript:

```
var TextEditor = function() {

};

/**
* @param {string} text
* @return {void}
*/
TextEditor.prototype.addText = function(text) {

};

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

```
* @return {number}
*/
TextEditor.prototype.deleteText = function(k) {

};

/**
* @param {number} k
* @return {string}
*/
TextEditor.prototype.cursorLeft = function(k) {

};

/**
* @param {number} k
* @return {string}
*/
TextEditor.prototype.cursorRight = function(k) {

};

/**
* Your TextEditor object will be instantiated and called as such:
* var obj = new TextEditor()
* obj.addText(text)
* var param_2 = obj.deleteText(k)
* var param_3 = obj.cursorLeft(k)
* var param_4 = obj.cursorRight(k)
*/
```

TypeScript:

```
class TextEditor {
constructor() {

}

addText(text: string): void {

}
```

```
deleteText(k: number): number {

}

cursorLeft(k: number): string {

}

cursorRight(k: number): string {

}
}

/**
* Your TextEditor object will be instantiated and called as such:
* var obj = new TextEditor()
* obj.addText(text)
* var param_2 = obj.deleteText(k)
* var param_3 = obj.cursorLeft(k)
* var param_4 = obj.cursorRight(k)
*/
```

C#:

```
public class TextEditor {

public TextEditor() {

}

public void AddText(string text) {

}

public int DeleteText(int k) {

}

public string CursorLeft(int k) {

}
```

```
public string CursorRight(int k) {

}
}

/**
* Your TextEditor object will be instantiated and called as such:
* TextEditor obj = new TextEditor();
* obj.AddText(text);
* int param_2 = obj.DeleteText(k);
* string param_3 = obj.CursorLeft(k);
* string param_4 = obj.CursorRight(k);
*/
```

C:

```
typedef struct {

} TextEditor;


TextEditor* textEditorCreate() {

}

void textEditorAddText(TextEditor* obj, char* text) {

}

int textEditorDeleteText(TextEditor* obj, int k) {

}

char* textEditorCursorLeft(TextEditor* obj, int k) {

}

char* textEditorCursorRight(TextEditor* obj, int k) {
```

```
}

void textEditorFree(TextEditor* obj) {

}

/**
* Your TextEditor struct will be instantiated and called as such:
* TextEditor* obj = textEditorCreate();
* textEditorAddText(obj, text);

* int param_2 = textEditorDeleteText(obj, k);

* char* param_3 = textEditorCursorLeft(obj, k);

* char* param_4 = textEditorCursorRight(obj, k);

* textEditorFree(obj);
*/
```

Go:

```
type TextEditor struct {

}


func Constructor() TextEditor {

}


func (this *TextEditor) AddText(text string) {

}


func (this *TextEditor) DeleteText(k int) int {

}
```

```
func (this *TextEditor) CursorLeft(k int) string {

}


func (this *TextEditor) CursorRight(k int) string {

}


/**
* Your TextEditor object will be instantiated and called as such:
* obj := Constructor();
* obj.AddText(text);
* param_2 := obj.DeleteText(k);
* param_3 := obj.CursorLeft(k);
* param_4 := obj.CursorRight(k);
*/
```

Kotlin:

```
class TextEditor() {

fun addText(text: String) {

}

fun deleteText(k: Int): Int {

}

fun cursorLeft(k: Int): String {

}

fun cursorRight(k: Int): String {

}

}

/**
```

```
* Your TextEditor object will be instantiated and called as such:
* var obj = TextEditor()
* obj.addText(text)
* var param_2 = obj.deleteText(k)
* var param_3 = obj.cursorLeft(k)
* var param_4 = obj.cursorRight(k)
*/
```

Swift:

```
class TextEditor {

init() {

}

func addText(_ text: String) {

}

func deleteText(_ k: Int) -> Int {

}

func cursorLeft(_ k: Int) -> String {

}

func cursorRight(_ k: Int) -> String {

}
}

/**
* Your TextEditor object will be instantiated and called as such:
* let obj = TextEditor()
* obj.addText(text)
* let ret_2: Int = obj.deleteText(k)
* let ret_3: String = obj.cursorLeft(k)
* let ret_4: String = obj.cursorRight(k)
*/
```

Rust:

```
struct TextEditor {

}


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

fn new() -> Self {

}

fn add_text(&self, text: String) {

}

fn delete_text(&self, k: i32) -> i32 {

}

fn cursor_left(&self, k: i32) -> String {

}

fn cursor_right(&self, k: i32) -> String {

}
}

/**
* Your TextEditor object will be instantiated and called as such:
* let obj = TextEditor::new();
* obj.add_text(text);
* let ret_2: i32 = obj.delete_text(k);
* let ret_3: String = obj.cursor_left(k);
* let ret_4: String = obj.cursor_right(k);
*/
```

Ruby:

```
class TextEditor
def initialize()

end


=begin
:type text: String
:rtype: Void
=end
def add_text(text)

end


=begin
:type k: Integer
:rtype: Integer
=end
def delete_text(k)

end


=begin
:type k: Integer
:rtype: String
=end
def cursor_left(k)

end


=begin
:type k: Integer
:rtype: String
=end
def cursor_right(k)

end
```

```
end

# Your TextEditor object will be instantiated and called as such:
# obj = TextEditor.new()
# obj.add_text(text)
# param_2 = obj.delete_text(k)
# param_3 = obj.cursor_left(k)
# param_4 = obj.cursor_right(k)
```

PHP:

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

}

/**
* @param String $text
* @return NULL
*/
function addText($text) {

}

/**
* @param Integer $k
* @return Integer
*/
function deleteText($k) {

}

/**
* @param Integer $k
* @return String
*/
function cursorLeft($k) {
```

```
}

/**
* @param Integer $k
* @return String
*/
function cursorRight($k) {

}
}

/**
* Your TextEditor object will be instantiated and called as such:
* $obj = TextEditor();
* $obj->addText($text);
* $ret_2 = $obj->deleteText($k);
* $ret_3 = $obj->cursorLeft($k);
* $ret_4 = $obj->cursorRight($k);
*/
```

Dart:

```
class TextEditor {

TextEditor() {

}

void addText(String text) {

}

int deleteText(int k) {

}

String cursorLeft(int k) {

}

String cursorRight(int k) {
```

```
}
}

/**
* Your TextEditor object will be instantiated and called as such:
* TextEditor obj = TextEditor();
* obj.addText(text);
* int param2 = obj.deleteText(k);
* String param3 = obj.cursorLeft(k);
* String param4 = obj.cursorRight(k);
*/
```

Scala:

```
class TextEditor() {

def addText(text: String): Unit = {

}

def deleteText(k: Int): Int = {

}

def cursorLeft(k: Int): String = {

}

def cursorRight(k: Int): String = {

}

}

/**
* Your TextEditor object will be instantiated and called as such:
* val obj = new TextEditor()
* obj.addText(text)
* val param_2 = obj.deleteText(k)
* val param_3 = obj.cursorLeft(k)
* val param_4 = obj.cursorRight(k)
*/
```

Elixir:

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

end

@spec add_text(text :: String.t) :: any
def add_text(text) do

end

@spec delete_text(k :: integer) :: integer
def delete_text(k) do

end

@spec cursor_left(k :: integer) :: String.t
def cursor_left(k) do

end

@spec cursor_right(k :: integer) :: String.t
def cursor_right(k) do

end
end

# Your functions will be called as such:
# TextEditor.init_()
# TextEditor.add_text(text)
# param_2 = TextEditor.delete_text(k)
# param_3 = TextEditor.cursor_left(k)
# param_4 = TextEditor.cursor_right(k)

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

Erlang:

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

-spec text_editor_add_text(Text :: unicode:unicode_binary()) -> any().
text_editor_add_text(Text) ->
.

-spec text_editor_delete_text(K :: integer()) -> integer().
text_editor_delete_text(K) ->
.

-spec text_editor_cursor_left(K :: integer()) -> unicode:unicode_binary().
text_editor_cursor_left(K) ->
.

-spec text_editor_cursor_right(K :: integer()) -> unicode:unicode_binary().
text_editor_cursor_right(K) ->
.


%% Your functions will be called as such:
%% text_editor_init_(),
%% text_editor_add_text(Text),
%% Param_2 = text_editor_delete_text(K),
%% Param_3 = text_editor_cursor_left(K),
%% Param_4 = text_editor_cursor_right(K),

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

Racket:

```
(define text-editor%
(class object%
(super-new)

(init-field)

; add-text : string? -> void?
(define/public (add-text text)
)
; delete-text : exact-integer? -> exact-integer?
```

```
(define/public (delete-text k)
)
; cursor-left : exact-integer? -> string?
(define/public (cursor-left k)
)
; cursor-right : exact-integer? -> string?
(define/public (cursor-right k)
)))

;; Your text-editor% object will be instantiated and called as such:
;; (define obj (new text-editor%))
;; (send obj add-text text)
;; (define param_2 (send obj delete-text k))
;; (define param_3 (send obj cursor-left k))
;; (define param_4 (send obj cursor-right k))
```

Solutions

C++ Solution:

```
/*
* Problem: Design a Text Editor
* Difficulty: Hard
* Tags: string, linked_list, stack
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

class TextEditor {
public:
TextEditor() {

}

void addText(string text) {

}
```

```
int deleteText(int k) {

}

string cursorLeft(int k) {

}

string cursorRight(int k) {

}
};

/**
* Your TextEditor object will be instantiated and called as such:
* TextEditor* obj = new TextEditor();
* obj->addText(text);
* int param_2 = obj->deleteText(k);
* string param_3 = obj->cursorLeft(k);
* string param_4 = obj->cursorRight(k);
*/
```

Java Solution:

```
/**
* Problem: Design a Text Editor
* Difficulty: Hard
* Tags: string, linked_list, stack
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

class TextEditor {

public TextEditor() {

}

public void addText(String text) {
```

```
}

public int deleteText(int k) {

}

public String cursorLeft(int k) {

}

public String cursorRight(int k) {

}
}

/**
* Your TextEditor object will be instantiated and called as such:
* TextEditor obj = new TextEditor();
* obj.addText(text);
* int param_2 = obj.deleteText(k);
* String param_3 = obj.cursorLeft(k);
* String param_4 = obj.cursorRight(k);
*/
```

Python3 Solution:

```
"""
Problem: Design a Text Editor
Difficulty: Hard
Tags: string, linked_list, stack

Approach: String manipulation with hash map or two pointers
Time Complexity: O(n) or O(n log n)
Space Complexity: O(1) to O(n) depending on approach
"""

class TextEditor:

def __init__(self):
```

```
def addText(self, text: str) -> None:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class TextEditor(object):

def __init__(self):


def addText(self, text):
"""
:type text: str
:rtype: None
"""


def deleteText(self, k):
"""
:type k: int
:rtype: int
"""


def cursorLeft(self, k):
"""
:type k: int
:rtype: str
"""


def cursorRight(self, k):
"""
:type k: int
:rtype: str
"""
```

```
# Your TextEditor object will be instantiated and called as such:
# obj = TextEditor()
# obj.addText(text)
# param_2 = obj.deleteText(k)
# param_3 = obj.cursorLeft(k)
# param_4 = obj.cursorRight(k)
```

JavaScript Solution:

```
/**
* Problem: Design a Text Editor
* Difficulty: Hard
* Tags: string, linked_list, stack
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/


var TextEditor = function() {

};

/**
* @param {string} text
* @return {void}
*/
TextEditor.prototype.addText = function(text) {

};

/**
* @param {number} k
* @return {number}
*/
TextEditor.prototype.deleteText = function(k) {

};

/**
```

```
* @param {number} k
* @return {string}
*/
TextEditor.prototype.cursorLeft = function(k) {

};

/**
* @param {number} k
* @return {string}
*/
TextEditor.prototype.cursorRight = function(k) {

};

/**
* Your TextEditor object will be instantiated and called as such:
* var obj = new TextEditor()
* obj.addText(text)
* var param_2 = obj.deleteText(k)
* var param_3 = obj.cursorLeft(k)
* var param_4 = obj.cursorRight(k)
*/
```

TypeScript Solution:

```
/**
* Problem: Design a Text Editor
* Difficulty: Hard
* Tags: string, linked_list, stack
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

class TextEditor {
constructor() {

}
```

```
addText(text: string): void {

}

deleteText(k: number): number {

}

cursorLeft(k: number): string {

}

cursorRight(k: number): string {

}
}

/**
* Your TextEditor object will be instantiated and called as such:
* var obj = new TextEditor()
* obj.addText(text)
* var param_2 = obj.deleteText(k)
* var param_3 = obj.cursorLeft(k)
* var param_4 = obj.cursorRight(k)
*/
```

C# Solution:

```
/*
* Problem: Design a Text Editor
* Difficulty: Hard
* Tags: string, linked_list, stack
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

public class TextEditor {

public TextEditor() {
```

```
}

public void AddText(string text) {

}

public int DeleteText(int k) {

}

public string CursorLeft(int k) {

}

public string CursorRight(int k) {

}
}

/**
* Your TextEditor object will be instantiated and called as such:
* TextEditor obj = new TextEditor();
* obj.AddText(text);
* int param_2 = obj.DeleteText(k);
* string param_3 = obj.CursorLeft(k);
* string param_4 = obj.CursorRight(k);
*/
```

C Solution:

```
/*
* Problem: Design a Text Editor
* Difficulty: Hard
* Tags: string, linked_list, stack
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/
```

```
typedef struct {

} TextEditor;


TextEditor* textEditorCreate() {

}

void textEditorAddText(TextEditor* obj, char* text) {

}

int textEditorDeleteText(TextEditor* obj, int k) {

}

char* textEditorCursorLeft(TextEditor* obj, int k) {

}

char* textEditorCursorRight(TextEditor* obj, int k) {

}

void textEditorFree(TextEditor* obj) {

}

/**
* Your TextEditor struct will be instantiated and called as such:
* TextEditor* obj = textEditorCreate();
* textEditorAddText(obj, text);

* int param_2 = textEditorDeleteText(obj, k);

* char* param_3 = textEditorCursorLeft(obj, k);

* char* param_4 = textEditorCursorRight(obj, k);
```

```
* textEditorFree(obj);
*/
```

Go Solution:

```
// Problem: Design a Text Editor
// Difficulty: Hard
// Tags: string, linked_list, stack
//
// Approach: String manipulation with hash map or two pointers
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(1) to O(n) depending on approach

type TextEditor struct {

}


func Constructor() TextEditor {

}


func (this *TextEditor) AddText(text string) {

}


func (this *TextEditor) DeleteText(k int) int {

}


func (this *TextEditor) CursorLeft(k int) string {

}


func (this *TextEditor) CursorRight(k int) string {
```

```
}


/**
* Your TextEditor object will be instantiated and called as such:
* obj := Constructor();
* obj.AddText(text);
* param_2 := obj.DeleteText(k);
* param_3 := obj.CursorLeft(k);
* param_4 := obj.CursorRight(k);
*/
```

Kotlin Solution:

```
class TextEditor() {

fun addText(text: String) {

}

fun deleteText(k: Int): Int {

}

fun cursorLeft(k: Int): String {

}

fun cursorRight(k: Int): String {

}

}

/**
* Your TextEditor object will be instantiated and called as such:
* var obj = TextEditor()
* obj.addText(text)
* var param_2 = obj.deleteText(k)
* var param_3 = obj.cursorLeft(k)
* var param_4 = obj.cursorRight(k)
```

```
*/
```

Swift Solution:

```
class TextEditor {

init() {

}

func addText(_ text: String) {

}

func deleteText(_ k: Int) -> Int {

}

func cursorLeft(_ k: Int) -> String {

}

func cursorRight(_ k: Int) -> String {

}
}

/**
* Your TextEditor object will be instantiated and called as such:
* let obj = TextEditor()
* obj.addText(text)
* let ret_2: Int = obj.deleteText(k)
* let ret_3: String = obj.cursorLeft(k)
* let ret_4: String = obj.cursorRight(k)
*/
```

Rust Solution:

```
// Problem: Design a Text Editor
// Difficulty: Hard
```

```
// Tags: string, linked_list, stack
//
// Approach: String manipulation with hash map or two pointers
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(1) to O(n) depending on approach

struct TextEditor {

}


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

fn new() -> Self {

}

fn add_text(&self, text: String) {

}

fn delete_text(&self, k: i32) -> i32 {

}

fn cursor_left(&self, k: i32) -> String {

}

fn cursor_right(&self, k: i32) -> String {

}
}

/**
* Your TextEditor object will be instantiated and called as such:
* let obj = TextEditor::new();
* obj.add_text(text);
```

```
* let ret_2: i32 = obj.delete_text(k);
* let ret_3: String = obj.cursor_left(k);
* let ret_4: String = obj.cursor_right(k);
*/
```

Ruby Solution:

```
class TextEditor
def initialize()

end


=begin
:type text: String
:rtype: Void
=end
def add_text(text)

end


=begin
:type k: Integer
:rtype: Integer
=end
def delete_text(k)

end


=begin
:type k: Integer
:rtype: String
=end
def cursor_left(k)

end


=begin
```

```
:type k: Integer
:rtype: String
=end
def cursor_right(k)

end



end

# Your TextEditor object will be instantiated and called as such:
# obj = TextEditor.new()
# obj.add_text(text)
# param_2 = obj.delete_text(k)
# param_3 = obj.cursor_left(k)
# param_4 = obj.cursor_right(k)
```

PHP Solution:

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

}

/**
* @param String $text
* @return NULL
*/
function addText($text) {

}

/**
* @param Integer $k
* @return Integer
*/
function deleteText($k) {

}
```

```
/**
* @param Integer $k
* @return String
*/
function cursorLeft($k) {

}

/**
* @param Integer $k
* @return String
*/
function cursorRight($k) {

}
}

/**
* Your TextEditor object will be instantiated and called as such:
* $obj = TextEditor();
* $obj->addText($text);
* $ret_2 = $obj->deleteText($k);
* $ret_3 = $obj->cursorLeft($k);
* $ret_4 = $obj->cursorRight($k);
*/
```

Dart Solution:

```
class TextEditor {

TextEditor() {

}

void addText(String text) {

}

int deleteText(int k) {
```

```
}

String cursorLeft(int k) {

}

String cursorRight(int k) {

}
}

/**
* Your TextEditor object will be instantiated and called as such:
* TextEditor obj = TextEditor();
* obj.addText(text);
* int param2 = obj.deleteText(k);
* String param3 = obj.cursorLeft(k);
* String param4 = obj.cursorRight(k);
*/
```

Scala Solution:

```
class TextEditor() {

def addText(text: String): Unit = {

}

def deleteText(k: Int): Int = {

}

def cursorLeft(k: Int): String = {

}

def cursorRight(k: Int): String = {

}

}
```

```
/**
* Your TextEditor object will be instantiated and called as such:
* val obj = new TextEditor()
* obj.addText(text)
* val param_2 = obj.deleteText(k)
* val param_3 = obj.cursorLeft(k)
* val param_4 = obj.cursorRight(k)
*/
```

Elixir Solution:

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

end

@spec add_text(text :: String.t) :: any
def add_text(text) do

end

@spec delete_text(k :: integer) :: integer
def delete_text(k) do

end

@spec cursor_left(k :: integer) :: String.t
def cursor_left(k) do

end

@spec cursor_right(k :: integer) :: String.t
def cursor_right(k) do

end
end

# Your functions will be called as such:
# TextEditor.init_()
```

```
# TextEditor.add_text(text)
# param_2 = TextEditor.delete_text(k)
# param_3 = TextEditor.cursor_left(k)
# param_4 = TextEditor.cursor_right(k)

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

Erlang Solution:

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

-spec text_editor_add_text(Text :: unicode:unicode_binary()) -> any().
text_editor_add_text(Text) ->
.

-spec text_editor_delete_text(K :: integer()) -> integer().
text_editor_delete_text(K) ->
.

-spec text_editor_cursor_left(K :: integer()) -> unicode:unicode_binary().
text_editor_cursor_left(K) ->
.

-spec text_editor_cursor_right(K :: integer()) -> unicode:unicode_binary().
text_editor_cursor_right(K) ->
.


%% Your functions will be called as such:
%% text_editor_init_(),
%% text_editor_add_text(Text),
%% Param_2 = text_editor_delete_text(K),
%% Param_3 = text_editor_cursor_left(K),
%% Param_4 = text_editor_cursor_right(K),

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

Racket Solution:

```
(define text-editor%
(class object%
(super-new)

(init-field)

; add-text : string? -> void?
(define/public (add-text text)
)
; delete-text : exact-integer? -> exact-integer?
(define/public (delete-text k)
)
; cursor-left : exact-integer? -> string?
(define/public (cursor-left k)
)
; cursor-right : exact-integer? -> string?
(define/public (cursor-right k)
)))

;; Your text-editor% object will be instantiated and called as such:
;; (define obj (new text-editor%))
;; (send obj add-text text)
;; (define param_2 (send obj delete-text k))
;; (define param_3 (send obj cursor-left k))
;; (define param_4 (send obj cursor-right k))
```