Problem 1622: Fancy Sequence

Problem Information

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

Problem Description

Write an API that generates fancy sequences using the

append

,

addAll

, and

multAll

operations.

Implement the

Fancy

class:

Fancy()

Initializes the object with an empty sequence.

void append(val)

Appends an integer

val

to the end of the sequence.

void addAll(inc)

Increments all existing values in the sequence by an integer

inc

.

void multAll(m)

Multiplies all existing values in the sequence by an integer

m

.

int getIndex(idx)

Gets the current value at index

idx

(0-indexed) of the sequence

modulo

10

9

+ 7

. If the index is greater or equal than the length of the sequence, return

-1

.

Example 1:

Input

["Fancy", "append", "addAll", "append", "multAll", "getIndex", "addAll", "append", "multAll", "getIndex", "getIndex", "getIndex"] [[], [2], [3], [7], [2], [0], [3], [10], [2], [0], [1], [2]]

Output

[null, null, null, null, null, 10, null, null, null, 26, 34, 20]

Explanation

Fancy fancy = new Fancy(); fancy.append(2); // fancy sequence: [2] fancy.addAll(3); // fancy sequence: [2+3] -> [5] fancy.append(7); // fancy sequence: [5, 7] fancy.multAll(2); // fancy sequence: [5*2, 7*2] -> [10, 14] fancy.getIndex(0); // return 10 fancy.addAll(3); // fancy sequence: [10+3, 14+3] -> [13, 17] fancy.append(10); // fancy sequence: [13, 17, 10] fancy.multAll(2); // fancy sequence: [13*2, 17*2, 10*2] -> [26, 34, 20] fancy.getIndex(0); // return 26 fancy.getIndex(1); // return 34 fancy.getIndex(2); // return 20

Constraints:

1 <= val, inc, m <= 100

0 <= idx <= 10

5

At most

10

5

calls total will be made to

append

,

addAll

,

multAll

, and

getIndex

.

Code Snippets

C++:

```
class Fancy {
public:
Fancy() {

}

void append(int val) {

}

void addAll(int inc) {

}

void multAll(int m) {

}

int getIndex(int idx) {
```

```
}
};

/**
* Your Fancy object will be instantiated and called as such:
* Fancy* obj = new Fancy();
* obj->append(val);
* obj->addAll(inc);
* obj->multAll(m);
* int param_4 = obj->getIndex(idx);
*/
```

Java:

```
class Fancy {

public Fancy() {

}

public void append(int val) {

}

public void addAll(int inc) {

}

public void multAll(int m) {

}

public int getIndex(int idx) {

}
}

/**
* Your Fancy object will be instantiated and called as such:
* Fancy obj = new Fancy();
* obj.append(val);
```

```
* obj.addAll(inc);
* obj.multAll(m);
* int param_4 = obj.getIndex(idx);
*/
```

Python3:

```
class Fancy:

def __init__(self):


def append(self, val: int) -> None:


def addAll(self, inc: int) -> None:


def multAll(self, m: int) -> None:


def getIndex(self, idx: int) -> int:



# Your Fancy object will be instantiated and called as such:
# obj = Fancy()
# obj.append(val)
# obj.addAll(inc)
# obj.multAll(m)
# param_4 = obj.getIndex(idx)
```

Python:

```
class Fancy(object):

def __init__(self):


def append(self, val):
"""
:type val: int
```

```
:rtype: None
"""


def addAll(self, inc):
"""
:type inc: int
:rtype: None
"""


def multAll(self, m):
"""
:type m: int
:rtype: None
"""


def getIndex(self, idx):
"""
:type idx: int
:rtype: int
"""



# Your Fancy object will be instantiated and called as such:
# obj = Fancy()
# obj.append(val)
# obj.addAll(inc)
# obj.multAll(m)
# param_4 = obj.getIndex(idx)
```

JavaScript:

```
var Fancy = function() {

};

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

```
* @return {void}
*/
Fancy.prototype.append = function(val) {

};

/**
* @param {number} inc
* @return {void}
*/
Fancy.prototype.addAll = function(inc) {

};

/**
* @param {number} m
* @return {void}
*/
Fancy.prototype.multAll = function(m) {

};

/**
* @param {number} idx
* @return {number}
*/
Fancy.prototype.getIndex = function(idx) {

};

/**
* Your Fancy object will be instantiated and called as such:
* var obj = new Fancy()
* obj.append(val)
* obj.addAll(inc)
* obj.multAll(m)
* var param_4 = obj.getIndex(idx)
*/
```

TypeScript:

```
class Fancy {
constructor() {

}

append(val: number): void {

}

addAll(inc: number): void {

}

multAll(m: number): void {

}

getIndex(idx: number): number {

}
}

/**
* Your Fancy object will be instantiated and called as such:
* var obj = new Fancy()
* obj.append(val)
* obj.addAll(inc)
* obj.multAll(m)
* var param_4 = obj.getIndex(idx)
*/
```

C#:

```
public class Fancy {

public Fancy() {

}

public void Append(int val) {

}
```

```
public void AddAll(int inc) {

}

public void MultAll(int m) {

}

public int GetIndex(int idx) {

}
}

/**
* Your Fancy object will be instantiated and called as such:
* Fancy obj = new Fancy();
* obj.Append(val);
* obj.AddAll(inc);
* obj.MultAll(m);
* int param_4 = obj.GetIndex(idx);
*/
```

C:

```
typedef struct {

} Fancy;


Fancy* fancyCreate() {

}

void fancyAppend(Fancy* obj, int val) {

}

void fancyAddAll(Fancy* obj, int inc) {
```

```
}

void fancyMultAll(Fancy* obj, int m) {

}

int fancyGetIndex(Fancy* obj, int idx) {

}

void fancyFree(Fancy* obj) {

}

/**
* Your Fancy struct will be instantiated and called as such:
* Fancy* obj = fancyCreate();
* fancyAppend(obj, val);

* fancyAddAll(obj, inc);

* fancyMultAll(obj, m);

* int param_4 = fancyGetIndex(obj, idx);

* fancyFree(obj);
*/
```

Go:

```
type Fancy struct {

}


func Constructor() Fancy {

}


func (this *Fancy) Append(val int) {
```

```
}


func (this *Fancy) AddAll(inc int) {

}


func (this *Fancy) MultAll(m int) {

}


func (this *Fancy) GetIndex(idx int) int {

}


/**
* Your Fancy object will be instantiated and called as such:
* obj := Constructor();
* obj.Append(val);
* obj.AddAll(inc);
* obj.MultAll(m);
* param_4 := obj.GetIndex(idx);
*/
```

Kotlin:

```
class Fancy() {

fun append(`val`: Int) {

}

fun addAll(inc: Int) {

}

fun multAll(m: Int) {

}
```

```
fun getIndex(idx: Int): Int {

}

}

/**
* Your Fancy object will be instantiated and called as such:
* var obj = Fancy()
* obj.append(`val`)
* obj.addAll(inc)
* obj.multAll(m)
* var param_4 = obj.getIndex(idx)
*/
```

Swift:

```
class Fancy {

init() {

}

func append(_ val: Int) {

}

func addAll(_ inc: Int) {

}

func multAll(_ m: Int) {

}

func getIndex(_ idx: Int) -> Int {

}
}
```

```
/**
* Your Fancy object will be instantiated and called as such:
* let obj = Fancy()
* obj.append(val)
* obj.addAll(inc)
* obj.multAll(m)
* let ret_4: Int = obj.getIndex(idx)
*/
```

Rust:

```
struct Fancy {

}


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

fn new() -> Self {

}

fn append(&self, val: i32) {

}

fn add_all(&self, inc: i32) {

}

fn mult_all(&self, m: i32) {

}

fn get_index(&self, idx: i32) -> i32 {

}
}
```

```
/**
* Your Fancy object will be instantiated and called as such:
* let obj = Fancy::new();
* obj.append(val);
* obj.add_all(inc);
* obj.mult_all(m);
* let ret_4: i32 = obj.get_index(idx);
*/
```

Ruby:

```
class Fancy
def initialize()

end


=begin
:type val: Integer
:rtype: Void
=end
def append(val)

end


=begin
:type inc: Integer
:rtype: Void
=end
def add_all(inc)

end


=begin
:type m: Integer
:rtype: Void
=end
def mult_all(m)
```

```
end


=begin
:type idx: Integer
:rtype: Integer
=end
def get_index(idx)

end



end

# Your Fancy object will be instantiated and called as such:
# obj = Fancy.new()
# obj.append(val)
# obj.add_all(inc)
# obj.mult_all(m)
# param_4 = obj.get_index(idx)
```

PHP:

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

}

/**
* @param Integer $val
* @return NULL
*/
function append($val) {

}

/**
* @param Integer $inc
* @return NULL
*/
```

```
function addAll($inc) {

}

/**
* @param Integer $m
* @return NULL
*/
function multAll($m) {

}

/**
* @param Integer $idx
* @return Integer
*/
function getIndex($idx) {

}
}

/**
* Your Fancy object will be instantiated and called as such:
* $obj = Fancy();
* $obj->append($val);
* $obj->addAll($inc);
* $obj->multAll($m);
* $ret_4 = $obj->getIndex($idx);
*/
```

Dart:

```
class Fancy {

Fancy() {

}

void append(int val) {

}
```

```
void addAll(int inc) {

}

void multAll(int m) {

}

int getIndex(int idx) {

}
}

/**
* Your Fancy object will be instantiated and called as such:
* Fancy obj = Fancy();
* obj.append(val);
* obj.addAll(inc);
* obj.multAll(m);
* int param4 = obj.getIndex(idx);
*/
```

Scala:

```
class Fancy() {

def append(`val`: Int): Unit = {

}

def addAll(inc: Int): Unit = {

}

def multAll(m: Int): Unit = {

}

def getIndex(idx: Int): Int = {

}
```

```
}

/**
* Your Fancy object will be instantiated and called as such:
* val obj = new Fancy()
* obj.append(`val`)
* obj.addAll(inc)
* obj.multAll(m)
* val param_4 = obj.getIndex(idx)
*/
```

Elixir:

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

end

@spec append(val :: integer) :: any
def append(val) do

end

@spec add_all(inc :: integer) :: any
def add_all(inc) do

end

@spec mult_all(m :: integer) :: any
def mult_all(m) do

end

@spec get_index(idx :: integer) :: integer
def get_index(idx) do

end
end

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

```
# Fancy.append(val)
# Fancy.add_all(inc)
# Fancy.mult_all(m)
# param_4 = Fancy.get_index(idx)

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

Erlang:

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

-spec fancy_append(Val :: integer()) -> any().
fancy_append(Val) ->
.

-spec fancy_add_all(Inc :: integer()) -> any().
fancy_add_all(Inc) ->
.

-spec fancy_mult_all(M :: integer()) -> any().
fancy_mult_all(M) ->
.

-spec fancy_get_index(Idx :: integer()) -> integer().
fancy_get_index(Idx) ->
.


%% Your functions will be called as such:
%% fancy_init_(),
%% fancy_append(Val),
%% fancy_add_all(Inc),
%% fancy_mult_all(M),
%% Param_4 = fancy_get_index(Idx),

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

Racket:

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

(init-field)

; append : exact-integer? -> void?
(define/public (append val)
)
; add-all : exact-integer? -> void?
(define/public (add-all inc)
)
; mult-all : exact-integer? -> void?
(define/public (mult-all m)
)
; get-index : exact-integer? -> exact-integer?
(define/public (get-index idx)
)))

;; Your fancy% object will be instantiated and called as such:
;; (define obj (new fancy%))
;; (send obj append val)
;; (send obj add-all inc)
;; (send obj mult-all m)
;; (define param_4 (send obj get-index idx))
```

Solutions

C++ Solution:

```
/*
* Problem: Fancy Sequence
* Difficulty: Hard
* Tags: tree, math
*
* Approach: DFS or BFS traversal
* Time Complexity: O(n) where n is number of nodes
* Space Complexity: O(h) for recursion stack where h is height
*/
```

```
class Fancy {
public:
Fancy() {

}

void append(int val) {

}

void addAll(int inc) {

}

void multAll(int m) {

}

int getIndex(int idx) {

}
};

/**
* Your Fancy object will be instantiated and called as such:
* Fancy* obj = new Fancy();
* obj->append(val);
* obj->addAll(inc);
* obj->multAll(m);
* int param_4 = obj->getIndex(idx);
*/
```

Java Solution:

```
/**
* Problem: Fancy Sequence
* Difficulty: Hard
* Tags: tree, math
*
* Approach: DFS or BFS traversal
```

```
* Time Complexity: O(n) where n is number of nodes
* Space Complexity: O(h) for recursion stack where h is height
*/

class Fancy {

public Fancy() {

}

public void append(int val) {

}

public void addAll(int inc) {

}

public void multAll(int m) {

}

public int getIndex(int idx) {

}
}

/**
* Your Fancy object will be instantiated and called as such:
* Fancy obj = new Fancy();
* obj.append(val);
* obj.addAll(inc);
* obj.multAll(m);
* int param_4 = obj.getIndex(idx);
*/
```

Python3 Solution:

```
"""
Problem: Fancy Sequence
Difficulty: Hard
```

```
Tags: tree, math

Approach: DFS or BFS traversal
Time Complexity: O(n) where n is number of nodes
Space Complexity: O(h) for recursion stack where h is height
"""

class Fancy:

def __init__(self):


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

Python Solution:

```
class Fancy(object):

def __init__(self):


def append(self, val):
"""
:type val: int
:rtype: None
"""


def addAll(self, inc):
"""
:type inc: int
:rtype: None
"""


def multAll(self, m):
"""
:type m: int
:rtype: None
```

```
"""



def getIndex(self, idx):
"""
:type idx: int
:rtype: int
"""



# Your Fancy object will be instantiated and called as such:
# obj = Fancy()
# obj.append(val)
# obj.addAll(inc)
# obj.multAll(m)
# param_4 = obj.getIndex(idx)
```

JavaScript Solution:

```
/**
* Problem: Fancy Sequence
* Difficulty: Hard
* Tags: tree, math
*
* Approach: DFS or BFS traversal
* Time Complexity: O(n) where n is number of nodes
* Space Complexity: O(h) for recursion stack where h is height
*/



var Fancy = function() {

};

/**
* @param {number} val
* @return {void}
*/
Fancy.prototype.append = function(val) {
```

```
};

/**
* @param {number} inc
* @return {void}
*/
Fancy.prototype.addAll = function(inc) {

};

/**
* @param {number} m
* @return {void}
*/
Fancy.prototype.multAll = function(m) {

};

/**
* @param {number} idx
* @return {number}
*/
Fancy.prototype.getIndex = function(idx) {

};

/**
* Your Fancy object will be instantiated and called as such:
* var obj = new Fancy()
* obj.append(val)
* obj.addAll(inc)
* obj.multAll(m)
* var param_4 = obj.getIndex(idx)
*/
```

TypeScript Solution:

```
/**
* Problem: Fancy Sequence
* Difficulty: Hard
* Tags: tree, math
```

```
*
* Approach: DFS or BFS traversal
* Time Complexity: O(n) where n is number of nodes
* Space Complexity: O(h) for recursion stack where h is height
*/

class Fancy {
constructor() {

}

append(val: number): void {

}

addAll(inc: number): void {

}

multAll(m: number): void {

}

getIndex(idx: number): number {

}
}

/**
* Your Fancy object will be instantiated and called as such:
* var obj = new Fancy()
* obj.append(val)
* obj.addAll(inc)
* obj.multAll(m)
* var param_4 = obj.getIndex(idx)
*/
```

C# Solution:

```
/*
* Problem: Fancy Sequence
```

```
* Difficulty: Hard
* Tags: tree, math
*
* Approach: DFS or BFS traversal
* Time Complexity: O(n) where n is number of nodes
* Space Complexity: O(h) for recursion stack where h is height
*/

public class Fancy {

public Fancy() {

}

public void Append(int val) {

}

public void AddAll(int inc) {

}

public void MultAll(int m) {

}

public int GetIndex(int idx) {

}
}

/**
* Your Fancy object will be instantiated and called as such:
* Fancy obj = new Fancy();
* obj.Append(val);
* obj.AddAll(inc);
* obj.MultAll(m);
* int param_4 = obj.GetIndex(idx);
*/
```

C Solution:

```c
/*
* Problem: Fancy Sequence
* Difficulty: Hard
* Tags: tree, math
*
* Approach: DFS or BFS traversal
* Time Complexity: O(n) where n is number of nodes
* Space Complexity: O(h) for recursion stack where h is height
*/




typedef struct {

} Fancy;


Fancy* fancyCreate() {

}

void fancyAppend(Fancy* obj, int val) {

}

void fancyAddAll(Fancy* obj, int inc) {

}

void fancyMultAll(Fancy* obj, int m) {

}

int fancyGetIndex(Fancy* obj, int idx) {

}

void fancyFree(Fancy* obj) {

}

/**
```

```
* Your Fancy struct will be instantiated and called as such:
* Fancy* obj = fancyCreate();
* fancyAppend(obj, val);

* fancyAddAll(obj, inc);

* fancyMultAll(obj, m);

* int param_4 = fancyGetIndex(obj, idx);

* fancyFree(obj);
*/
```

Go Solution:

```
// Problem: Fancy Sequence
// Difficulty: Hard
// Tags: tree, math
//
// Approach: DFS or BFS traversal
// Time Complexity: O(n) where n is number of nodes
// Space Complexity: O(h) for recursion stack where h is height

type Fancy struct {

}


func Constructor() Fancy {

}


func (this *Fancy) Append(val int) {

}


func (this *Fancy) AddAll(inc int) {

}
```

```
func (this *Fancy) MultAll(m int) {

}


func (this *Fancy) GetIndex(idx int) int {

}


/**
* Your Fancy object will be instantiated and called as such:
* obj := Constructor();
* obj.Append(val);
* obj.AddAll(inc);
* obj.MultAll(m);
* param_4 := obj.GetIndex(idx);
*/
```

Kotlin Solution:

```
class Fancy() {

fun append(`val`: Int) {

}

fun addAll(inc: Int) {

}

fun multAll(m: Int) {

}

fun getIndex(idx: Int): Int {

}
```

```
}

/**
* Your Fancy object will be instantiated and called as such:
* var obj = Fancy()
* obj.append(`val`)
* obj.addAll(inc)
* obj.multAll(m)
* var param_4 = obj.getIndex(idx)
*/
```

Swift Solution:

```
class Fancy {

init() {

}

func append(_ val: Int) {

}

func addAll(_ inc: Int) {

}

func multAll(_ m: Int) {

}

func getIndex(_ idx: Int) -> Int {

}
}

/**
* Your Fancy object will be instantiated and called as such:
* let obj = Fancy()
* obj.append(val)
```

```
* obj.addAll(inc)
* obj.multAll(m)
* let ret_4: Int = obj.getIndex(idx)
*/
```

Rust Solution:

```
// Problem: Fancy Sequence
// Difficulty: Hard
// Tags: tree, math
//
// Approach: DFS or BFS traversal
// Time Complexity: O(n) where n is number of nodes
// Space Complexity: O(h) for recursion stack where h is height

struct Fancy {

}



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

fn new() -> Self {

}

fn append(&self, val: i32) {

}

fn add_all(&self, inc: i32) {

}

fn mult_all(&self, m: i32) {

}
```

```
fn get_index(&self, idx: i32) -> i32 {

}
}

/**
* Your Fancy object will be instantiated and called as such:
* let obj = Fancy::new();
* obj.append(val);
* obj.add_all(inc);
* obj.mult_all(m);
* let ret_4: i32 = obj.get_index(idx);
*/
```

Ruby Solution:

```
class Fancy
def initialize()

end


=begin
:type val: Integer
:rtype: Void
=end
def append(val)

end


=begin
:type inc: Integer
:rtype: Void
=end
def add_all(inc)

end
```

```
=begin
:type m: Integer
:rtype: Void
=end
def mult_all(m)

end


=begin
:type idx: Integer
:rtype: Integer
=end
def get_index(idx)

end



end

# Your Fancy object will be instantiated and called as such:
# obj = Fancy.new()
# obj.append(val)
# obj.add_all(inc)
# obj.mult_all(m)
# param_4 = obj.get_index(idx)
```

PHP Solution:

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

}

/**
* @param Integer $val
* @return NULL
*/
function append($val) {
```

```
}

/**
* @param Integer $inc
* @return NULL
*/
function addAll($inc) {

}

/**
* @param Integer $m
* @return NULL
*/
function multAll($m) {

}

/**
* @param Integer $idx
* @return Integer
*/
function getIndex($idx) {

}
}

/**
* Your Fancy object will be instantiated and called as such:
* $obj = Fancy();
* $obj->append($val);
* $obj->addAll($inc);
* $obj->multAll($m);
* $ret_4 = $obj->getIndex($idx);
*/
```

Dart Solution:

```
class Fancy {
```

```
Fancy() {

}

void append(int val) {

}

void addAll(int inc) {

}

void multAll(int m) {

}

int getIndex(int idx) {

}
}

/**
* Your Fancy object will be instantiated and called as such:
* Fancy obj = Fancy();
* obj.append(val);
* obj.addAll(inc);
* obj.multAll(m);
* int param4 = obj.getIndex(idx);
*/
```

Scala Solution:

```
class Fancy() {

def append(`val`: Int): Unit = {

}

def addAll(inc: Int): Unit = {

}
```

```
def multAll(m: Int): Unit = {

}

def getIndex(idx: Int): Int = {

}

}

/**
* Your Fancy object will be instantiated and called as such:
* val obj = new Fancy()
* obj.append(`val`)
* obj.addAll(inc)
* obj.multAll(m)
* val param_4 = obj.getIndex(idx)
*/
```

Elixir Solution:

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

end

@spec append(val :: integer) :: any
def append(val) do

end

@spec add_all(inc :: integer) :: any
def add_all(inc) do

end

@spec mult_all(m :: integer) :: any
def mult_all(m) do
```

```
end

@spec get_index(idx :: integer) :: integer
def get_index(idx) do

end
end

# Your functions will be called as such:
# Fancy.init_()
# Fancy.append(val)
# Fancy.add_all(inc)
# Fancy.mult_all(m)
# param_4 = Fancy.get_index(idx)

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

Erlang Solution:

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

-spec fancy_append(Val :: integer()) -> any().
fancy_append(Val) ->
.

-spec fancy_add_all(Inc :: integer()) -> any().
fancy_add_all(Inc) ->
.

-spec fancy_mult_all(M :: integer()) -> any().
fancy_mult_all(M) ->
.

-spec fancy_get_index(Idx :: integer()) -> integer().
fancy_get_index(Idx) ->
.
```

```
%% Your functions will be called as such:
%% fancy_init_(),
%% fancy_append(Val),
%% fancy_add_all(Inc),
%% fancy_mult_all(M),
%% Param_4 = fancy_get_index(Idx),

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

Racket Solution:

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

(init-field)

; append : exact-integer? -> void?
(define/public (append val)
)
; add-all : exact-integer? -> void?
(define/public (add-all inc)
)
; mult-all : exact-integer? -> void?
(define/public (mult-all m)
)
; get-index : exact-integer? -> exact-integer?
(define/public (get-index idx)
)))

;; Your fancy% object will be instantiated and called as such:
;; (define obj (new fancy%))
;; (send obj append val)
;; (send obj add-all inc)
;; (send obj mult-all m)
;; (define param_4 (send obj get-index idx))
```