Problem 3 – Clojure Parser

Chavdar is a JavaScript developer, but he wants to learn some new technology. Technology that will change the way he thinks! He heard from friends that functional programming is really cool, especially the new rockstar there – **Clojure.**

Clojure's syntax is really simple and easy to read (except the trailing ")"). Here are some examples:

(+ 1 2) //Returns 3
(+ 5 2 7 1) //Returns 15
(As you can see you can add multiple numbers, and **you set the operation at the beginning of the sequence**)
`(- 4 2) //2`
`(- 4) //Return 4`
`(/ 2) //Returns 2`
	`(* 5 7) //35`
	`(/ 10 3) //Returns 3`
	(**The division returns number without the remainder**)
	`(/ 10 3 2) //1 (10/3)/2`
		`(/ 5 0)// Returns` **`Division by zero! At Line:N`**
		(Where **N** is the line number)
		You Can define functions
		`(def NewFunc 5)`
		`//Now` **`NewFunc`** `is equal to 5`

Chavdar couldn't install leiningen(famous Clojure interpretator) on his Windows to run his Clojure code, so he decided to make some really basic Clojure Parser. The parser should execute lines of basic Clojure commands (defining function and some math operations). Also It has to print the result of the last given line of code. Each line starts with "("and ends with ")".

NOTE: There could be more than one whitespace between the characters. For example
`(  +  4  1  ) //Also has to return 5`

Also you can use old functions in the definitions of the new one. The parser should run code in this format:

```
(def myFunc 5) // myFunc = 5
(def myNewFunc (+  myFunc  myFunc 2)) //myNewFunc = 12
(def MyFunc (* 2  myNewFunc)) //MyFunc = 24,myFunc = 5(Names are CaseSensitive)
(/  MyFunc  myFunc) //Now the parser should return 4
```

NOTE: There will be no more than 1 nested function in the definition of a new one.
Example: Command can be (def func (+ 23 4 8 4 5 7)), but it **won't be** (def func (+ 23 4 (* 2 2) 4 5 7))

NOTE: There will be no nested functions in the last line. Only math operation and a sequence of numbers and functions.
Example: Last line can be (+ 23 4 8 myFunc 5 7), but it **won't be** (+ 23 4 (* 2 myFunc) 4 newFunc 7))

You are given an array of strings (commands). Execute all the commands and **print the result only from the last line!**

- If you meet a function in a command it'll be always defined in some of the lines before!
- "- 5" is not valid number but "-5" is.
- Function's names are case sensitive.
- Functions cannot be overwritten (and there'll be no attempts to do that).
- Each string will be a valid Clojure command (Except of the case when we are trying to divide a number by zero)

Write method **Solve** that accepts the commands as array and prints the result of the last command.

Input

The method **Solve** accepts an array of strings. (Example: arr=[“command1”,”command2”,”command3”])

Output

Your method should return a single line - the result of the last command

Example code

```
function Solve(params) {
	var answer = 0;
	// Your code here...
	return answer;
}
```

Constraints

- Array size will be between 1 and 1000 elements.
- Each element of the Array will be string containing valid command.
- Allowed working time for your program: 0.2 seconds. Allowed memory: 16 MB.

Examples (each line represents an element (string) from the only argument of Solve)

Example input	**Example output**

`(def func 10)`
`(def newFunc (+  func 2))`
`(def sumFunc (+ func func newFunc 0 0 0))`
`(* sumFunc 2)`
	64

Example input	**Example output**

`(def func (+ 5 2))`
`(def func2 (/  func 5 2 1 0))`
`(def func3 (/ func 2))`
`(+ func3 func)`
	Division by zero! At Line:2