2018.1.13 - Stack processor

Because you are working with a team that developed a new stack processor architecture, you decided to help in the development phase by creating a software simulator for the processor in question. The simulator will receive, as a real processor, a sequence of instructions that will be executed and the final data will be displayed as the output.

The processor is able to execute the following instructions, which you must interpret in your program:

- **iload <val>** - add the <val> value to the top of the stack;
- **iadd** - two values are removed from the stack, summed, and the result is added back to the stack;
- **imul** - two values are removed from the stack, multiplied, and the result is added back to the stack;
- **dup** – duplicate the stack head by adding the stack head value once more to the stack.

If for the execution of an instruction there are not enough values on the stack (two for **iadd** and **imul** and one for **dup**) the program will display the phrase **Exception: line <index>** where index is the line of the instruction that could not be executed, and the program will end. The stack is initially empty.

Requirement

Given a sequence of instructions as above, run them and display the number of values on the stack and those values.

Input data

From the keyboard (standard input stream, *stdin*) read from the first line a number **n**, the number of instructions. Each of the following **n** lines contains an instruction as described above.

Output data

The program will display in the console (stream standard output, *stdout*) on the first line the number of items remaining on the stack and on the following n lines, these elements, from stack head to stack tail, each on a separate line.

ATTENTION to the compliance to the problem requirements: the display of results must be done EXACTLY as required! In other words, on the standard output stream there will be nothing displayed in addition to the problem requirements; following the automatic evaluation, any supplemental character displayed, or any display different than the requirements, will produce an eroneous result and will lead to the „Reject" of the solution.

Restrictions and specifications

1. $0 < n \leq 1000$
2. The values on the stack are always signed 32bit integer values.
3. **Warning:** According to the chosen programming language, the file containing the code must have one of the extensions .c, .cpp, .java, or .m. The web editor does not add automatically these extensions and the lack of the extensions leads to the impossibility of program compilation!

4. **Warning:** The source file must be named by the candidate as: <name>.<ext> where name is the family name (last name) of the candidate and the extension is the one chosen according to the previous warning. Attention to the restrictions imposed by the Java language regarding the class name and the file name!

Examples

entry	exit	Explanations

`3`
`iload 5`
`iload 3`
`iadd`
	`1`
	`8`
		Three instructions are executed, the stack is initially empty.
		• **iload 5** will load 5 on the stack.
		• **iload 3** will load 3 on the stack, which is now the top.
		• **iadd** will remove the last two elements (3 and 5) from the stack and their sum (8) will be added back on the stack.
		So finally we have one item on the stack: 8

`6`
`iload 100`
`dup`
`iload 1`
`iadd`
`imul`
`iload 0`
	`2`
	`0`
	`10100`
		Six instructions are executed, the stack is initially empty.
		• **iload 100** will load 100 on the stack.
		• **dup** duplicates the top of the stack (100) so now the stack contains two values, both 100.
		• **iload 1** will load 1 on the stack, the stack thus comprises of three elements 100, 100, 1 (head)
		• **iadd** removes values 1 and 100 and the sum is added to the stack 100, 101 (head)
		• **imul** will eliminate two values from the top of the stack and their product value is added back: 10100
		• **iload 0** will add 0 to the top of the stack: 10100, 0 (head)
		So finally we have two values on the stack: 0 10100

`5`
`iload 12`
`iload 13`
`imul`
`iadd`
`iload 10`
`iadd`
	`Exception: line 4`	Five instructions are executed, the stack is initially empty.
		• **iload 12** loads 12 on the stack.
		• **iload 13** loads 13 on the stack: 12, 13 (head)
		• **imul** multiplies the first two values (12 and 13): 156 (head) – now the stack only has one value
		• **iadd** should multiply the first two values from the top of the stack, but the stack has only one item, so it displays exception on line 4, which is the index of the instruction that could not be executed, and the program stops.

Working time: 120 minutes