Task: ROZ
Accounting

XXVI OI, Stage II, Day trial. Source file `roz.*` Available memory: 10 MB. *12.02.2019*

Byteasar has become the chief accountant at the Bytecom company. This means handling invoices, which are complex financial documents. However, in Byteasar's experience, only the so called invoice total matters in each invoice. To avoid confusion, we shall call the invoice total the invoice value and associate it with the invoice itself. To ease his work, which involves manipulating invoice values arranged in a list, Byteasar has commissioned you with developing a computer accounting system. It has to support the following operations on the invoice list:

- appending a new invoice at the end of the list,
- correcting the value of the i-th invoice from the end of the list,
- reporting the sum of values of the last i invoices in the list.

The system has to allow executing an arbitrary number of such operations, where the correction and summation operations never pertain more than the last m invoices on the list — we assume that the ones preceding them have been settled already: no corrections are possible, and asking about them makes little sense.

Write a module that provides the following functions to facilitate accounting. Pay particular attention to the **available memory limit** and the fact that you are not supposed to provide the `main` function. Memory used by grading program does **not** count towards memory limit.

Communication

Your module has to provide the following functions:

- `inicjuj`(m)
  This function will be called only once, at the beginning of the program execution. You may use it to learn the value of m ($1 \leq m$) and initialize any data structures.
  - **C++:** `void inicjuj(int m);`
  - **Python:** `def inicjuj(m)`

- `dodaj`(k)
  This function should append a new invoice with value k ($-10^9 \leq k \leq 10^9$) at the end of the invoice list.
  - **C++:** `void dodaj(int k);`
  - **Python:** `def dodaj(k)`

- `koryguj`(i, k)
  This function should correct the value of the i-th invoice from the end of the list ($1 \leq i \leq m$) by adding k ($-10^9 \leq k \leq 10^9$) to it. In particular, $i = 1$ corresponds to correcting the value of the invoice which was appended last. You may assume that there are at least i invoices on the list when the function is called.
  - **C++:** `void koryguj(int i, int k);`
  - **Python:** `def koryguj(i, k)`

- `suma`(i)
  This function should return the sum of values of the last i ($1 \leq i \leq m$) invoices on the list. If less then i invoices were appended, it should return the sum of all invoice values.
  - **C++:** `long long suma(int i);`
  - **Python:** `def suma(i)`

Your program **cannot** read any data, neither from standard input nor any files. Likewise, it **cannot** write to any files nor the standard output. It can write to the standard diagnostic output (`stderr`) – remember though that this takes (precious) time.

Grading

The set of tests consists of the following subsets. Within each subset, there may be several unit tests. The number of operations in a single unit test will not exceed 10 000 000.

Time limits for each subset are published in SIO.

Subset	Condition	Score
1	$m \leq 50$	10
2	$m \leq 100\,000$	33
3	$m \leq 250\,000$	35
4	$m \leq 500\,000$	11
5	$m \leq 1\,000\,000$	11

Sample program execution

Operation	Result	Explanation
`inicjuj(3)`		$m = 3$
`dodaj(-6)`		append an invoice with value -6
`suma(1)`	-6	report the value of the last invoice
`dodaj(5)`		append an invoice with value 5
`koryguj(2, 10)`		correct the second invoice from the end by changing its value to $-6 + 10 = 4$
`suma(3)`	9	report the sum of values of all invoices
`suma(1)`	5	report the value of the last invoice

Experiments

The `dlazaw` directory contains files that will let you test formal correctness of your solution: sample grading programs (`rozgrader.cpp` and `rozgrader.py`). Note that these are merely for the sake of checking correctness of interaction with the grading program and differ from those which will eventually evaluate your solution. To have the grading program evaluate your library, put the library in a file named `roz.cpp` or `roz.py`. Initially, these contain incorrect sample solutions which should help you understand the interface. To get an executable file from the grading program and your library, type:

```
make
```

As a result, a file `rozCPP.e` or `rozPy.e` will be created, depending on the language the library was written in. Compilation in C++ requires the file `rozinc.h`, which is also provided in appropriate directories.

Thus created grading program reads from the standard input a specially formatted test description, calls appropriate functions from your library, and prints the results to the standard output.

The test description format is as follows: The first line contains a single integer m. The second line contains a single integer q, specifying the number of functions to be called, `inicjuj` excluded. Next, q lines follow, each of which contains a single character `d`, `k` or `s` as well as one (a) or two(a, b) integers. The character specifies the function to be called: `d` for `dodaj(`a`)`, `k` for `koryguj(`a`, `b`)` and `s` for `suma(`a`)`. The `inicjuj` function will be called immediately after reading the very first line.

Keep in mind though that the provided sample grading program does not check if the input data is in the correct format, nor whether aforementioned conditions are satisfied.

The directory with tests also contains an input file `roz0.in`, which corresponds to the sample program execution discussed earlier. To execute the grading program on this sample input, use the following commands:

- **C++:** `./rozCPP.e < roz0.in`
- **Python:** `./rozPy.e < roz0.in`

The values returned by calls of `suma` will be printed to the standard output. The correct output for the sample program execution can be found in the file `roz0.out`.

Sample grading tests:

`1ocen:` $m = 250$, the total number of calls of the functions `dodaj`, `koryguj`, and `suma` is 600. First, `dodaj` is called 500 times with successive numbers from -250 to 249. Next, for i ranging from 1 to 100 the following alternating calls are made:

- `koryguj(`i`, `$2i$`)` for odd i,
- `suma(`i`)` for even i.

`2ocen:` $m = 1000$, the total number of calls of the functions `dodaj`, `koryguj`, and `suma` is 6000. First, `dodaj` is called 1000 times with successive numbers from 0 to 999. Next, for i ranging from 1 to 5000 the following alternating calls are made:

- `koryguj(`$1 + i \bmod m$`, `$2i$`)` for $i \bmod 3 = 1$,
- `suma(`$1 + i \bmod m$`)` for $i \bmod 3 = 2$,
- `dodaj(`i`)` for $i \bmod 3 = 0$.