

Использовать стандартную библиотеку (`vector`, `ArrayList`, `LinkedList`, и т. п.) не разрешается.

Задача А. Стек на векторе

Имя входного файла: `stack1.in`
Имя выходного файла: `stack1.out`

В этой задаче обязательно использовать самостоятельно написанный вектор.

Реализуйте работу стека. Для каждой операции изъятия элемента выведите ее результат.

На вход программе подаются строки, содержащие команды. Каждая строка содержит одну команду. Команда — это либо “+ N”, либо “-”. Команда “+ N” означает добавление в стек числа N , по модулю не превышающего 10^9 . Команда “-” означает изъятие элемента из стека. Гарантируется, что не происходит извлечения из пустого стека.

Формат входного файла

В первой строке входного файла содержится количество команд — M ($1 \leq M \leq 10^6$). Каждая последующая строка исходного файла содержит ровно одну команду.

Формат выходного файла

Выведите числа, которые удаляются из стека, по одному в каждой строке. Гарантируется, что изъятий из пустого стека не производится.

Пример

<code>stack1.in</code>	<code>stack1.out</code>
6	10
+ 1	1234
+ 10	
-	
+ 2	
+ 1234	
-	

Задача В. Стек на списке

Имя входного файла: `stack2.in`
Имя выходного файла: `stack2.out`

В этой задаче обязательно использовать самостоятельно написанный односвязный список. Реализуйте работу стека. Для каждой операции изъятия элемента выведите ее результат.

На вход программе подаются строки, содержащие команды. Каждая строка содержит одну команду. Команда — это либо “+ N”, либо “-”. Команда “+ N” означает добавление в стек числа N , по модулю не превышающего 10^9 . Команда “-” означает изъятие элемента из стека. Гарантируется, что не происходит извлечения из пустого стека.

Формат входного файла

В первой строке входного файла содержится количество команд — M ($1 \leq M \leq 10^6$). Каждая последующая строка исходного файла содержит ровно одну команду.

Формат выходного файла

Выведите числа, которые удаляются из стека, по одному в каждой строке. Гарантируется, что изъятий из пустого стека не производится.

Пример

<code>stack2.in</code>	<code>stack2.out</code>
6	10
+ 1	1234
+ 10	
-	
+ 2	
+ 1234	
-	

Задача С. Очередь на векторе

Имя входного файла: `queue1.in`
Имя выходного файла: `queue1.out`

В этой задаче обязательно использовать самостоятельно написанный вектор. Вы можете использовать любую стратегию увеличения/уменьшения размера массива, но размер массива должен быть не больше Cn , где n — количество элементов в очереди, для некоторого $C > 1$.

Реализуйте работу очереди. Для каждой операции изъятия элемента выведите ее результат.

На вход программе подаются строки, содержащие команды. Каждая строка содержит одну команду. Команда — это либо “+ N”, либо “-”. Команда “+ N” означает добавление в очередь числа N , по модулю не превышающего 10^9 . Команда “-” означает изъятие элемента из очереди.

Формат входного файла

В первой строке содержится количество команд — M ($1 \leq M \leq 10^6$). В последующих строках содержатся команды, по одной в каждой строке.

Формат выходного файла

Выведите числа, которые удаляются из очереди, по одному в каждой строке. Гарантируется, что извлечения из пустой очереди не производится.

Пример

<code>queue1.in</code>	<code>queue1.out</code>
4	1
+ 1	10
+ 10	
-	
-	

Задача D. Очередь на списке

Имя входного файла: `queue2.in`
Имя выходного файла: `queue2.out`

В этой задаче обязательно использовать самостоятельно написанный связный список.

Реализуйте работу очереди. Для каждой операции изъятия элемента выведите ее результат.

На вход программе подаются строки, содержащие команды. Каждая строка содержит одну команду. Команда — это либо “+ N”, либо “-”. Команда “+ N” означает добавление в очередь числа N , по модулю не превышающего 10^9 . Команда “-” означает изъятие элемента из очереди.

Формат входного файла

В первой строке содержится количество команд — M ($1 \leq M \leq 10^6$). В последующих строках содержатся команды, по одной в каждой строке.

Формат выходного файла

Выведите числа, которые удаляются из очереди, по одному в каждой строке. Гарантируется, что извлечения из пустой очереди не производится.

Пример

<code>queue2.in</code>	<code>queue2.out</code>
4	1
+ 1	10
+ 10	
-	
-	

Задача Е. Правильная скобочная последовательность

Имя входного файла: `brackets.in`
Имя выходного файла: `brackets.out`

Входной файл содержит несколько строк, каждая из которых содержит последовательность символов '(', ')', '[' и ']'. Выясните, является ли она правильной скобочной последовательностью с двумя типами скобок.

Используйте реализацию стека из задачи 1 или 2, на ваше усмотрение.

Пример

<code>brackets.in</code>	<code>brackets.out</code>
<code>()()</code>	<code>YES</code>
<code>([])</code>	<code>YES</code>
<code>([)]</code>	<code>NO</code>
<code>(([])</code>	<code>NO</code>
<code>)()</code>	<code>NO</code>

Задача F. Постфиксная запись

Имя входного файла: postfix.in
Имя выходного файла: postfix.out

В постфиксной записи (или обратной польской записи) операция записывается после двух операндов. Например, сумма двух чисел A и B записывается как $A B +$. Запись $B C + D *$ обозначает привычное нам $(B+C)*D$, а запись $A B C + D * +$ означает $A+(B+C)*D$. Достоинство постфиксной записи в том, что она не требует скобок и дополнительных соглашений о приоритете операторов для своего чтения.

Дано выражение в обратной польской записи. Определите его значение.

Формат входного файла

В единственной строке записано выражение в постфиксной записи, содержащее однозначные числа и операции $+$, $-$, $*$. Строка содержит не более 100 чисел и операций.

Формат выходного файла

Необходимо вывести значение записанного выражения. Гарантируется, что результат выражения, а также результаты всех промежуточных вычислений по модулю меньше 2^{31} .

Пример

postfix.in	postfix.out
8 9 + 1 7 - *	-102

Задача G. Минимум в очереди

Имя входного файла: `queuemin2.in`
Имя выходного файла: `queuemin2.out`

Вам дан массив $x[1 \dots n]$ и число m . Для всех i от 1 до $n - m + 1$ найдите минимум из чисел $x[i], x[i + 1], \dots, x[i + m - 1]$. Вам требуется вывести сумму всех таких минимумов.

Используйте очередь с поддержкой операции “минимум в очереди”.

Формат входного файла

Первая строка входного файла содержит три числа: n , m и k ($1 \leq n \leq 30\,000\,000$, $1 \leq m \leq n$, $2 \leq k \leq \min(n, 1000)$). Вторая строка содержит три целых числа: a , b и c ($-2^{31} \leq a, b, c \leq 2^{31} - 1$). Третья строка содержит k целых чисел: $x[1], x[2], \dots, x[k]$ ($-2^{31} \leq x[i] \leq 2^{31} - 1$).

Остальные элементы массива вычисляются по следующей формуле: $x[i] = f(a \cdot x[i - 2] + b \cdot x[i - 1] + c)$. Здесь $f(y)$ возвращает такое число $-2^{31} \leq z \leq 2^{31} - 1$, что $y - z$ делится на 2^{32} .

Формат выходного файла

Выведите одно число — сумму минимумов на всех отрезках длины m .

Пример

<code>queuemin2.in</code>	<code>queuemin2.out</code>
10 3 2 1 1 0 0 1	33
1000000 15 5 283471207 23947205 3 17625384 939393931 1838388 912740247 290470294	-1879262596173354

Задача N. Интерпретатор языка Quack

Имя входного файла: quack.in
Имя выходного файла: quack.out

Язык Quack — забавный язык, который фигурирует в задаче G с IPSC 2004. В этой задаче вам требуется написать интерпретатор языка Quack.

Виртуальная машина, на которой исполняется программа на языке Quack имеет внутри себя очередь, содержащую целые числа по модулю 65536 (`get` в описании операций означает извлечение из очереди, `put` — добавление в очередь). Кроме того, у виртуальной машины есть 26 регистров, которые обозначаются буквами от `a` до `z`.

<code>+</code>	Сложение: <code>get x, get y, put (x+y) modulo 65536</code>
<code>-</code>	Вычитание: <code>get x, get y, put (x-y) modulo 65536</code> .
<code>*</code>	Умножение: <code>get x, get y, put (x*y) modulo 65536</code> .
<code>/</code>	Целочисленное деление: <code>get x, get y, put x div y</code> . (будем считать, что $0 \text{ div } 0 = 0$)
<code>%</code>	Взятие по модулю: <code>get x, get y, put x modulo y</code> . (будем считать, что $0 \text{ mod } 0 = 0$)
<code>>[register]</code>	Положить в регистр: <code>get x, установить значение [register]</code> в <code>x</code> .
<code><[register]</code>	Взять из регистра: <code>put значение [register]</code> .
<code>P</code>	Напечатать: <code>get x, вывести x</code> в стандартный поток вывода и перевести строку.
<code>P[register]</code>	Вывести значение регистра <code>[register]</code> в стандартный поток вывода и перевести строку.
<code>C</code>	Вывести как символ: <code>get x, вывести символ с ASCII кодом $x \text{ modulo } 256$</code> в стандартный поток вывода.
<code>C[register]</code>	Вывести регистр как символ: вывести символ с ASCII кодом $x \text{ modulo } 256$ (где <code>x</code> — значение регистра <code>[register]</code>) в стандартный поток вывода.
<code>:[label]</code>	Метка: эта строка программы имеет метку <code>[label]</code> .
<code>J[label]</code>	Переход на строку с меткой <code>[label]</code> .
<code>Z[register][label]</code>	Переход если 0: если значение регистра <code>[register]</code> равно нулю, выполнение программы продолжается с метки <code>[label]</code> .
<code>E[register1][register2][label]</code>	Переход если равны: если значения регистров <code>[register1]</code> и <code>[register2]</code> равны, исполнение программы продолжается с метки <code>[label]</code> .
<code>G[register1][register2][label]</code>	Переход если больше: если значение регистра <code>[register1]</code> больше, чем значение регистра <code>[register2]</code> , исполнение программы продолжается с метки <code>[label]</code> .
<code>Q</code>	Завершить работу программы. Работа также завершается, если выполнение доходит до конца программы.
<code>[number]</code>	Просто число во входном файле — <code>put</code> это число.

Формат входного файла

Входной файл содержит корректную синтаксически программу на языке Quack. Известно, что программа завершает работу не более чем за 100 000 шагов.

Формат выходного файла

Выведите содержимого стандартного потока вывода виртуальной машины в выходной файл.

Пример

quack.in	quack.out
100 0 :start >a Zaend <a <a 1 + - >b <b Jstart :end P	5050

Второй пример подразумевает UNIX-переводы строки (один символ с кодом 10).

Дискретный анализ
Лабораторная работа по линейным структурам данных, 2010 год

quack.in	quack.out
58	58
49	49
10	10
62	62
97	97
10	10
80	80
97	97
10	10
90	90
97	97
50	50
10	10
60	60
97	97
10	10
74	74
49	49
10	10
58	58
50	50
10	10
48	48
10	10
58	58
51	51
10	10
62	62
97	97
10	10
90	90
97	97
52	52
10	10
67	67
97	97
10	10
74	74
51	51
10	10
58	58
52	52
10	10
0	0
:1	:1
>a	>a
Pa	Pa
Za2	Za2
<a	<a
J1	J1
:2	:2
0	0
:3	:3
>a	>a
Za4	Za4
Ca	Ca
J3	J3
:4	:4