

С. Корпоративные закупки

	Все языки	GNU C++20 10.2
Ограничение времени	2 секунды	1 секунда
Ограничение памяти	512Mb	512Mb
Ввод	стандартный ввод или input.txt	
Вывод	стандартный вывод или output.txt	

Стартап Алисы Селезневой и Зелибобы привлек к себе внимание крупных инвесторов. Часть полученных от инвесторов денег было решено потратить на обновление офиса — новая мебель, оргтехника и другие прикольные штуки.

Алиса и Зелибоба выдвинули 5 критериев — товар должен удовлетворять всем данным критериям, чтобы его закупили в обновленный офис.

- «Наименование товара содержит подстроку **в любом регистре**» (критерий 'NAME\_CONTAINS');
- «Цена **больше либо равна** чем» (критерий 'PRICE\_GREATER\_THAN');
- «Цена **меньше либо равна** чем» (критерий 'PRICE\_LESS\_THAN');
- «Товар поступил в продажу **не позднее** чем» (критерий 'DATE\_BEFORE');
- «Товар поступил в продажу **не ранее** чем» (критерий 'DATE\_AFTER');

Закупаться было решено в Выньдекс.Рынке. Для такого крупного клиента Выньдекс.Рынок предоставил стартапу персонального менеджера — да-да, именно вас.

Первым делом вам необходимо из имеющегося списка товаров на складе выбрать все товары, удовлетворяющие выданным Алисой и Зелибобой критериям.

Формат ввода

Общее описание формата входных данных:  
Первая строка входных данных содержит список всех имеющихся на складе Выньдекс.Рынка товаров в формате JSON.

Следующие 5 строк имеют вид  $q_i v_i$  — фильтр и соответствующее ему актуальное значение.

Подробное описание формата списка товаров

Гарантии по формату JSON:

- нет запятых после последнего элемента массива;
- все имена полей и строки обернуты в двойные кавычки.

Обозначим количество товаров в списке через  $N$ . Гарантируется, что  $0 \leq N \leq 1000$ .

Каждый товар в списке содержит следующую информацию (порядок полей не является фиксированным):

- целое число  $id$  ( $0 \leq id \leq 2^{31} - 1$ ) — уникальный идентификатор. Гарантируется, что идентификаторы всех товаров попарно различны;
- строка  $name$  ( $1 \leq |name| \leq 100$ ) — наименование. Гарантируется, что наименование содержит только строчные и заглавные латинские буквы, а так же пробел;
- целое число  $price$  ( $0 \leq price \leq 2^{31} - 1$ ) — цена;
- строка  $date$  в формате «dd.MM.yyyy» ( $01.01.1970 \leq date \leq 31.12.2070$ ) — дата поступления в продажу.

Подробное описание формата фильтров

Гарантируется, что:

- все  $q_i$  различны между собой;
- $q_i$  является строкой из множества (NAME\_CONTAINS, PRICE\_GREATER\_THAN, PRICE\_LESS\_THAN, DATE\_BEFORE, DATE\_AFTER);
- в фильтре 'NAME\_CONTAINS'  $v_i$  представляет из себя строку ( $1 \leq |v_i| \leq 100$ ), содержащую только строчные и заглавные латинские буквы;
- в фильтрах 'PRICE\_GREATER\_THAN' и 'PRICE\_LESS\_THAN'  $v_i$  представляет из себя целое число ( $0 \leq v_i \leq 2^{31} - 1$ );
- в фильтрах 'DATE\_BEFORE' и 'DATE\_AFTER'  $v_i$  представляет из себя строку в формате «dd.MM.yyyy» ( $01.01.1970 \leq v_i \leq 31.12.2070$ ).

Формат вывода

Выведите в формате JSON список товаров, удовлетворяющих всем указанным во входных данных критериям. Каждый товар должен быть выведен ровно один раз в отсортированном по возрастанию *id* порядке.

Выводить JSON допустимо как с дополнительными отступами и переводами строк, так и в одну строку.

Имена полей необходимо выводить в двойных кавычках.

Допустимо выводить запятую после последнего поля объекта или последнего элемента массива.

Каждый товар должен содержать информацию, аналогичную информации из входных данных:

- целое число *id* — уникальный идентификатор;
- строка *name* — наименование;
- целое число *price* — цена;
- строка *date* в формате «dd.MM.yyyy» — дата поступления в продажу.

## Пример

Ввод

```
[{"id": 1, "name": "Asus notebook", "price": 1564, "date": "23.09.2021"}, {"price": 2500, "id": 3, "date": "05.06.2020", "name": "Dell notebook", "price_less_than": 2400, "date_after": "23.09.2021", "name_contains": "pods", "price_greater_than": 2200, "date_before": "02.01.2022"}]
```

## Примечания

При написании решения на Java можно выбрать компилятор «Java 8 + json-simple». В этом случае вы сможете воспользоваться библиотекой [json-simple](#) для парсинга и сериализации JSON.

При написании решения на C++ можно подключить `#include "json.hpp"` для использования библиотеки [json](#) для парсинга и сериализации JSON.

### Рассмотрим тестовый пример.

В нем представлено 5 товаров:

- `"id": 1, "name": "Asus notebook", "price": 1564, "date": "23.09.2021"`
- `"id": 2, "name": "EaRPods", "price": 2200, "date": "01.01.2022"`
- `"id": 3, "name": "Keyboardpods", "price": 2500, "date": "05.06.2020"`
- `"id": 4, "name": "Dell notebook", "price": 2300, "date": "23.09.2021"`
- `"id": 5, "name": "Airpods", "price": 2300, "date": "23.09.2021"`

и следующие критерии:

- название включает подстроку `pods` в любом регистре;
- цена находится в промежутке  $2200 \leq price \leq 2400$ ;
- дата поступления в продажу находится в промежутке  $23.09.2021 \leq date \leq 02.01.2022$ .

Только товары с идентификаторами 2 и 5 удовлетворяют всем критериям:

- Товар с идентификатором 1 не удовлетворяет критериям имени (нет заданной подстроки) и цены (слишком низкая);
- Товар с идентификатором 3 не удовлетворяет критериям цены (слишком высокая) и даты (слишком ранняя);
- Товар с идентификатором 4 не удовлетворяет только критериям имени (нет заданной подстроки).

Обратите внимание, что выводить необходимо товары в порядке возрастания идентификаторов (заметьте, что во входных данных товар с идентификатором 5 стоит раньше товара с идентификатором 2).

Язык Python 3.7 (PyPy 7.3.3)

Набрать здесь

Отправить файл

```
1 from datetime import datetime
2 from operator import itemgetter
3 import json
4
5
6 market = json.loads(input())
7 param_1 = input().split(' ')
8 param_2 = input().split(' ')
9 param_3 = input().split(' ')
10 param_4 = input().split(' ')
11 param_5 = input().split(' ')
12
13 requirements = {
14     'market': market,
15     param_1[0]: param_1[1],
16     param_2[0]: param_2[1],
17     param_3[0]: param_3[1],
18     param_4[0]: param_4[1],
19     param_5[0]: param_5[1],
20 }
21
22 requirements['PRICE_LESS_THAN'] = int(requirements['PRICE_LESS_THAN'])
23 requirements['PRICE_GREATER_THAN'] = int(requirements['PRICE_GREATER_THAN'])
24 requirements['DATE_AFTER'] = datetime.strptime(requirements['DATE_AFTER'], '%d.%m.%Y').date()
25 requirements['DATE_BEFORE'] = datetime.strptime(requirements['DATE_BEFORE'], '%d.%m.%Y').date()
26 requirements['NAME_CONTAINS'] = requirements['NAME_CONTAINS'].lower()
27
28
29 result = []
30
31 for article in requirements['market']:
32     name_low = article['name'].lower()
33     name = article['name']
34     price = int(article['price'])
35     prod_date = datetime.strptime(article['date'], '%d.%m.%Y').date()
36     id_num = int(article['id'])
37
38
```

Отправить

Предыдущая

Следующая