Задача 1. Минимальная дата

Источник:	базовая*
Имя входного файла:	`input.txt`
Имя выходного файла:	`output.txt`
Ограничение по времени:	1 секунда
Ограничение по памяти:	разумное

Требуется реализовать функцию для поиска самой ранней даты среди заданного массива дат. Даты включают в себя не только день, но и точное время. При помощи реализованной функции нужно решить тестовую задачу.

Каждая дата должна представляться структурой:

```
typedef struct DateTime_s {
    int year, month, day;
    int hours, minutes, seconds;
} DateTime;
```

Функция для поиска самой ранней (минимальной) даты должна иметь сигнатуру:

```
DateTime min(const DateTime *arr, int cnt);
```

Здесь `arr` — указатель на первый элемент массива дат, а `cnt` — длина массива.

Формат входных данных

В первой строке содержится целое число N — количество дат в файле ($2 \leqslant N \leqslant 50\,000$). В каждой из следующих N строк описана одна дата в виде шести целых чисел: `year`, `month`, `day`, `hours`, `minutes`, `seconds`. Гарантируется, что все даты корректны, а год лежит в пределах от 1 до 5 000 включительно.

Формат выходных данных

Нужно вывести самую раннюю дату среди записанных в файле дат в том же формате, в котором даты записываются во входных данных.

Пример

input.txt	output.txt
5 2018 8 12 23 44 13 2018 9 1 9 0 0 2019 1 1 0 0 0 2018 2 13 13 1 7 2018 8 26 8 20 11	2018 2 13 13 1 7

Задача 2. Имена и возрасты

Источник:	базовая
Имя входного файла:	`input.txt`
Имя выходного файла:	`output.txt`
Ограничение по времени:	1 секунда
Ограничение по памяти:	разумное

Есть набор рисунков с подписями автором в стиле “Зоя 17 лет”. Нужно реализовать функцию для сбора статистики по именам и возрастам авторов.

Каждая подпись должна представляться структурой:

```
typedef struct Label_s {
    char name[16];   //имя автора (заканчивается нулём)
    int age;         //возраст автора (сколько лет)
} Label;
```

Статистика имён должна представляться структурой:

```
typedef struct NameStats_s {
    int cntTotal;    //сколько всего подписей
    int cntLong;     //сколько подписей с именами длиннее 10 букв
} NameStats;
```

Статистика возрастов должна представляться структурой:

```
typedef struct AgeStats_s {
    int cntTotal;    //сколько всего подписей
    int cntAdults;   //сколько подписей взрослых (хотя бы 18 лет)
    int cntKids;     //сколько подписей детей (меньше 14 лет)
} AgeStats;
```

Функция для вычисления статистик должна иметь сигнатуру:

```
void calcStats(const Label *arr, int cnt, NameStats *oNames, AgeStats *oAges);
```

Здесь `oNames` и `oAges` — адреса структур, в которые нужно записать результат.

Формат входных данных

В первой строке содержится целое число N — количество подписей в файле ($1 \leqslant N \leqslant 1\,000$). Каждая подпись записана в виде “`[имя] [возраст] let`”.

Все имена не длиннее 15 символов, возрасты целые, положительные, не больше 5 000.

Формат выходных данных

Нужно вывести статистики `NameStats` и `AgeStats` в формате, приведённом в примере.

Пример

input.txt	output.txt
7 Zoya 17 let Kirill 5 let Ivan 1 let Vasiliy 15 let Tutankhamun 3360 let Innokentiy 21 let Dozdraperma 70 let	names: total = 7 names: long = 2 ages: total = 7 ages: adult = 3 ages: kid = 2

Задача 3. Разделить на слова

Источник:	базовая*
Имя входного файла:	`input.txt`
Имя выходного файла:	`output.txt`
Ограничение по времени:	1 секунда
Ограничение по памяти:	разумное

Требуется реализовать функции для выделения слов из заданной строки. Слова отделены друг от друга символами-разделителями, которые передаются в строку как параметр.

Сигнатура функций должна быть такой:

```
typedef struct Tokens_s {
    int num;          //количество слов в строке
    char **arr;       //массив слов (каждый элемент -- строка, т.е. char*)
} Tokens;
//tokens: структура, в которую нужно положить результаты
//str: строка, в которой нужно искать слова
//delims: все символы-разделители в виде строки
void tokensSplit(Tokens *tokens, const char *str, const char *delims);
//удаляет все ресурсы внутри tokens
void tokensFree(Tokens *tokens);
```

Память для массива слов `tokens->arr` следует выделять динамически ровно на столько элементов, сколько слов в строке. Поскольку заранее это количество неизвестно, то нужно запускать алгоритм два раза: первый раз, чтобы узнать сколько слов, и второй раз, чтобы записать слова в массив.

Ваша реализация вышеописанных функцию должна работать согласно следующим договорённостям:

1. Вызывающий гарантирует, что параметр `tokens` **не** нулевой (для обеих функций) и указывает на структуру `Tokens`.
2. Если при вызове `tokensSplit` указатель `tokens->arr` нулевой, то внутри функции нужно только посчитать количестов слов и записать его в `tokens->num`.
3. Если при вызове `tokensSplit` указатель `tokens->arr` **не** нулевой, то он должен указывать на массив, в который точно войдут все слова. В этом случае реализация функции должна записать в `tokens->num` количество слов, а в `tokens->arr[i]` записать i-ое слово, самостоятельно выделив под него память с помощью `malloc`.
4. Функция `tokensFree` должна удалять массив слов и сами строки-слова с помощью `free`. При этом программа должна работать корректно, даже если эту функцию случайно вызовут два или три раза подряд.

Таким образом, вызывающий может завести структуру `tokens`, потом определить количество слов первым вызовом `tokensSplit`, затем выделить память на массив `tokens->arr`, и, наконец, найти все слова вторым вызовом `tokensSplit`.

С помощью этих функций нужно решить тестовую задачу. Дана одна строка длиной до 10^6, состоящая из букв латинского алфавита и знаков препинания четырёх типов: точка, запятая, точка с запятой, двоеточие. Нужно найти слова в этой строке, состоящие из букв, и вывести их в файл в таком же формате, как показано в примере.

Пример

input.txt	output.txt
..ko,.Privet:kreved,.,;ko:;,.	4 ko Privet kreved ko

Комментарий

Следует выводить слова ровно в том порядке, в котором они встречаются в строке.

Задача 4. XOR double

Источник:	основная
Имя входного файла:	`input.txt`
Имя выходного файла:	`output.txt`
Ограничение по времени:	1 секунда
Ограничение по памяти:	разумное

Требуется взять заданное число X типа `double`, "прохогить" его с заданным 64-битным целым числом M, и вывести результат как `double`. "Прохогить" означает: обратить в битовом представлении X все биты, для которых бит с тем же номером в битовом представлении M равен единице.

В этой задаче предполагается little-endian порядок байтов и общепринятое 8-байтовое представление `double`.

Формат входных данных

В первой строке записано целое число N — количество тестов ($1 \leqslant N \leqslant 1\,000$). В остальных N строках записаны тесты, по одному в строке.

Каждый тест описан в формате: `"P/Q xor M"`. Здесь целые числа `P` и `Q` — числитель и знаменатель дроби, задающей вещественное число X ($0 \leqslant P \leqslant 100$, $1 \leqslant Q \leqslant 100$), а `M` — шестнадцатеричное целое число M ровно из шестнадцати цифр. В записи `M` сначала идут старшие цифры, потом младшие (как обычно у людей записываются числа).

Совет: Шестнадцитеричное число можно читать при помощи формата `"%x"` так же, как мы считаем десятичные числа форматом `"%d"`. Если нужно прочитать 64-битное число, то нужно дописать две буквы `ll` перед последней буквой формата.

Формат выходных данных

Для каждого теста выведите в отдельной строке ($X \operatorname{xor} M$) как вещественное число типа `double`.

Ваши ответы должны быть верны с относительной точностью 10^{-14}. Рекомендуется использовать формат `"%0.15g"` при выводе ответа.

Пример

input.txt	output.txt
10	-0
0/1 xor 8000000000000000	1
1/1 xor 0000000000000000	-1
1/1 xor 8000000000000000	0
1/1 xor 3ff0000000000000	2
1/1 xor 7ff0000000000000	0.5
1/1 xor 0010000000000000	1.625
1/1 xor 000a000000000000	-0.428571428571429
3/7 xor 8000000000000000	-2.90689205178751e-054
3/7 xor 8b0abc0000000000	0.428571428570292
3/7 xor 000000000000d000	

Пояснение к примеру

В первом тесте $X = 0/1$, а в заданной маске M установлен только старший бит. В представлении нуля в `double` все биты нулевые, после операции xor старший бит становится

единичным, однако число по-прежнему остаётся нулевым (получается так называемый “отрицательный ноль”).

Во втором тесте число $X = 1/1$ равно единице, а маска M вся нулевая. Значит xor ничего не меняет и результат получается тоже равен единице.

В третьем и восьмом тестах в маске только старший бит единичный. Он в представлении `double` отвечает за знак числа, так что в этих тестах число X меняет знак.

В предпоследнем тесте число $X = 3/7$, его битовое представление выглядит как `3fdb6db6db6db6db` в шестнадцатеричном виде. Когда мы xor-им с заданной маской, получается представление `b4d1d1b6db6db6db`. Если проинтерпретировать эти данные как `double`, то получается число `-2.90689205178751e-054`.

Задача 5. Распечатать список

Источник:	основная
Имя входного файла:	`input.txt`
Имя выходного файла:	`output.txt`
Ограничение по времени:	1 секунда
Ограничение по памяти:	разумное

Дан односвязный список, который хранится в массиве. Массив состоит из узлов, в каждом узле лежит какое-то вещественное значение и индекс следующего элемента списка в том же массиве. Нужно распечатать вещественные значения всех узлов в порядке их следования в связном списке.

Формат входных данных

В первой строке входного файла содержится два целых числа: N — количество узлов в массиве и F — индекс первого элемента связного списка ($0 \leqslant F < N \leqslant 100\,000$).

В каждой k-ой из оставшихся N строк записано содержимое k-ого элемента массива. Сначала записывается значение (вещественное число с тремя или меньше знаками после точки, которое по модулю не превышает 10^4), а затем индекс следующего элемента.

Все индексы элементов нумеруются с нуля. В последнем узле связного списка индекс следующего элемента равен -1.

Формат выходных данных

В выходной файл необходимо вывести N строк, по одному вещественному числу в каждой строке — значения всех узлов в порядке их следования в связном списке.

Вещественные числа нужно выводить хотя бы с тремя знаками после точки, например, используя формат `"%0.3lf"`.

Пример

input.txt	output.txt
5 3 4.283 2 2.718 4 5.0 -1 1.111 1 3.141 0	1.111 2.718 3.141 4.283 5.000

Пояснение к примеру

В примере порядок элементов получается: 3 (1.111) -> 1 (2.718) -> 4 (3.141) -> 0 (4.283) -> 2 (5.0).

Задача 6. Односвязный список

Источник:	основная
Имя входного файла:	`input.txt`
Имя выходного файла:	`output.txt`
Ограничение по времени:	1 секунда
Ограничение по памяти:	разумное

Требуется реализовать односвязный список на массиве. В каждом узле списка хранится строковое значение и индекс следующего узла. Дано начальное состояние списка, а также последовательность операций двух видов: добавление и удаление узла. Нужно выполнить все операции, и после этого вывести значения всех узлов списка в порядке их следования в цепи.

У каждого узла списка есть индекс в едином массиве, в котором всё хранится. В начальном состоянии списка есть N узлов, имеющих индексы от 0 до $N-1$ в порядке их задания. При добавлении нового узла он дописывается в конец массива. То есть первый добавленный узел имеет индекс N, второй добавленный — $N+1$, и так далее.

Нужно выполнять операции двух видов:

0. *Добавление узла.* При этом указывается индекс узла, после которого нужно вставить новый узел, и строковое значение для нового узла. Если индекс равен -1, то узел надо вставить в начало списка (перед самым первым элементом). После выполнения операции нужно вывести индекс нового узла (они назначаются по порядку, см. выше).
1. *Удаление узла.* При этом указывается индекс узла, и удалить нужно тот узел, который идёт сразу после него. Гарантируется, что узел с указанным индексом не последний. Если указан индекс -1, значит нужно удалить самый первый элемент списка (гарантируется, что он есть). После выполнения операции нужно вывести строковое значение удалённого узла.

В задаче используется «мультитест»: в одном входном файле записано много отдельных тестов.

Формат входных данных

В первой строке файла записано одно целое число T — количество тестов в файле. Далее в файле идут тесты (T штук) подряд, один за другим.

Первая строка теста начинается с трёх целых чисел: N — изначальное количество узлов в связном списке, F — индекс первого элемента списка и Q — количество операций, которые нужно выполнить ($0 \leqslant F < N \leqslant 10^5$, $0 \leqslant Q \leqslant 10^5$).

Затем идёт N строк, в которых описываются узлы связного списка в порядке увеличения индекса. Описание каждого узла состоит из его строкового значения и индекса следующего узла в списке (или -1, если следующего нет).

Наконец, в тесте идут Q строк, которые описывают операции над списком. В каждой строке сначала записан тип операции: 0 — добавление, 1 — удаление. Затем указан индекс узла, после которого нужно вставить/удалить узел. Если описывается операция вставки узла, то в конце также задано строковое значение нового узла.

У каждого узла строковое значение имеет длину от 1 до 7 символов включительно, и состоит из произвольных печатаемых символов ASCII (такие символы имеют код от 33 до 126 включительно).

Сумма N по всем тестам не превышает 10^5, аналогично для суммы всех Q.

Формат выходных данных

Для каждого теста нужно вывести следующее:

1. результаты выполнения операций (Q строк);
2. строка "==="(три знака равенства);
3. строковые значения всех узлов списка после выполнения операций (в порядке их следования в списке);
4. снова строка "===".

Пример

input.txt	output.txt
3 5 3 0 4.283 2 2.718 4 5.0 -1 1.111 1 3.141 0 1 0 5 zero -1 0 -1 one 0 -1 two 0 1 three 0 0 four 1 2 4 2 2 45 1 %drill# 3 &qw6: 0 *a-+r -1 1 1 0 2 \num\	=== 1.111 2.718 3.141 4.283 5.0 === 1 2 3 4 one === two three zero four === *a-+r 4 === &qw6: \num\ 45 %drill# ===

Пояснение к примеру

В примере три теста.

Первый тест полностью совпадает с примером к задаче “Распечатать список”: дано 5 узлов и 0 операций. Т.к. операций нет, в ответе записана сразу строка "===". Потом записан ответ как в задаче “Распечатать список” и ещё одна строка “===”.

Во втором тесте изначально есть только один узел. Заметим, что в каждом узле значение равно индексу, записанному по-английски: `zero`, `one`, `two`, `three`, `four`.

Первые четыре операции задают вставку элементов: сначала вставляется два узла в начало, получается список `two`, `one`, `zero`. Затем вставляется узел после узла `one` и ещё один узел после узла `zero`, получается список `two`, `one`, `three`, `zero`, `four`. Последняя операция удаления: удаляется узел, который стоит после узла `two`, то есть узел `one`.

Задача 7. Числа Фибоначчи

Источник:	основная
Имя входного файла:	`input.txt`
Имя выходного файла:	`output.txt`
Ограничение по времени:	1 секунда
Ограничение по памяти:	разумное

Последовательность чисел Фибоначчи определяется следующим образом. Первое и второе числа Фибоначчи равны единице, а каждое следующее число Фибоначчи равно сумме двух предыщущих. Вот первые числа Фибоначчи: 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, ...

Во входном файле задано одно целое число N в диапазоне $1 \leqslant N \leqslant 2\,000$. Нужно вычислить и распечатать N-ое число Фибоначчи.

Поскольку это число может быть довольно большим (более 400 цифр), то вам необходимо собственноручно реализовать десятичную арифметику.

Для этого рекомендуется использовать следующую структуру длинного числа:

```
typedef struct LongNum_s {
    int len;          //сколько цифр в числе
    int arr[500];     //массив десятичных цифр числа
} LongNum;
```

Далее вам следует реализовать алгоритм сложения длинных чисел «в столбик», как учили в школе, а также написать функцию распечатывания длинного числа. Тогда вы сможете вычислить нужное число Фибоначчи простым циклом по N.

Пример

input.txt	output.txt
12	144

Комментарий

Рекомендуется хранить цифры в массиве `arr` в обратном принятому у людей порядке:
`arr[0]` — это единицы,
`arr[1]` — десятки,
а `arr[len-1]` — это ведущая цифра.
Кстати, такой порядок «по-умному» называется little-endian.

Задача 8. Факториал

Источник:	повышенной сложности
Имя входного файла:	`input.txt`
Имя выходного файла:	`output.txt`
Ограничение по времени:	1 секунда
Ограничение по памяти:	разумное

Факториал от N (обозначается как $N!$) — это произведение всех целых чисел от 1 до N включительно.

Во входном файле задано одно целое число N в диапазоне $1 \leqslant N \leqslant 1\,000$. Нужно вычислить и распечатать факториал от N.

На этот раз вам понадобится обеспечить 3 000 цифр в структуре длинного числа, чтобы влез результат. Для вычисления ответа можно перебирать все k по порядку от 1 до N, и каждый раз домножать текущее длинное число на него.

Здесь нужно также реализовать умножение «в столбик», однако можно полагать множитель одной цифрой (даже когда он больше десяти). В некотором смысле, эта операция называется умножением длинного числа (заданного массивом цифр) на короткое (заданное одним числом), и сама по себе она реализуется без вложенных циклов.

Пример

input.txt	output.txt
10	3628800

Задача 9. sql join

Источник:	повышенной сложности
Имя входного файла:	`input.txt`
Имя выходного файла:	`output.txt`
Ограничение по времени:	2 секунды
Ограничение по памяти:	разумное

Многие приложения используют базы данных для постоянного хранения информации.

На сегодняшний день наиболее популярны реляционные базы данных. В реляционной базе данные представляются в виде таблиц. Каждая таблица состоит из произвольного набора записей, каждая запись занимает одну строку таблицы. Запись можно воспринимать как структуру языка C: в ней есть фиксированный набор полей фиксированного типа. Каждый столбец таблицы содержит значение одного конкретного поля для всех записей. Таким образом, в таблице строки задают отдельные записи, а столбцы — поля этих записей.

Для извлечения и фильтрации данных из реляционной базы данных чаще всего используют язык SQL. При этом нередко пользователю базы данных нужно составить сборный отчёт по нескольким таблицам. В таком случае можно использовать операции соединения (join) таблиц. В данной задаче предлагается реализовать операцию Inner Join для двух конкретных таблиц.

Первая таблица содержит биографии известных актёров кино. Она была создана следующей командой SQL:

```
CREATE TABLE ActorBio (
  Name varchar(30),
  BirthYear int,
  Country varchar(10)
);
```

Первый столбец называется `Name` и хранит имя актёра. Во втором записан год рождения как целое число. А третьем столбце записана страна, в которой жил актёр.

Вторая таблица содержит информацию о том, какой актёр в каких фильмах играл. Она была создана командой:

```
CREATE TABLE ActorInMovie (
  ActorName varchar(30),
  MovieName varchar(20)
);
```

Первый столбец содержит имя актёра `ActorName`, а второй — название кино, в котором этот актёр играл роль.

От вас требуется реализовать следующий SQL-запрос:

```
SELECT *
FROM ActorBio INNER JOIN ActorInMovie
ON ActorBio.Name = ActorInMovie.ActorName
```

Результатом этой операции является одна таблица, в которой пять полей: первые три поля взяты из таблицы `ActorBio`, а последние два — из `ActorInMovie`.

Механизм выполнения операции следующий:

1. Перебираем все пары записей A и B, где A взята из таблицы `ActorBio`, а B — из таблицы `ActorInMovie`.

2. Для каждой пары проверяем условие соединения: что имя актёра `Name` в записи A совпадает с именем актёра `ActorName` в записи B.
3. Если условие выполнено, то конкатенируем записи A и B и полученную запись с пятью полями добавляем в таблицу-результат.

Чтобы было проще понять, как работает операция, крайне рекомендуется посмотреть первый пример к задаче.

Формат входных данных

В первой строке входного файла записано целое число N — количество записей в таблице `ActorBio` ($1 \leqslant N \leqslant 10^5$). Далее идёт N строк, в которых записаны записи таблицы `ActorBio`. Затем записано целое число M — количество записей в таблице `ActorInMovie` ($1 \leqslant M \leqslant 10^5$). Далее идёт M строк, в которых записаны записи таблицы `ActorInMovie`.

Для каждой записи в файле записываются все её поля через пробел, в том порядке, в котором они определены. Все записи кроме года рождения строковые: они окружены символом двойной кавычки (ASCII 34) с обеих сторон. Каждое строковое значение непустое и может содержать в себе любые печатные символы ASCII (коды от 32 до 126 включительно) кроме символа двойной кавычки. Год рождения записан как целое число. Имена актёров не длиннее 30 символов, названия стран не длиннее 10 символов, названия фильмов не длиннее 20 символов.

Формат выходных данных

Требуется вывести ровно K строк: записи, получившиеся в таблице-результате после соединения. Каждая строка должна описывать одну запись с пятью полями, в том же формате, в котором эти записи записаны во входных данных. Порядок записей в выходном файле может быть любой.

Гарантируется, что $N \cdot M \leqslant 10^5$.

Пример

input.txt

```
8
"Peter Falk" 1927 "USA"
"Oleg Tabakov" 1935 "USSR"
"Andrei Mironov" 1941 "USSR"
"Arnold Schwarzenegger" 1947 "USA"
"Jean Reno" 1948 "France"
"Sharon Stone" 1958 "USA"
"Tom Cruise" 1962 "USA"
"Ryoko Hirosue" 1980 "Japan"
12
"Sharon Stone" "Basic Instinct"
"Jean Reno" "Mission: Impossible"
"Arnold Schwarzenegger" "Total Recall"
"Tom Cruise" "Mission: Impossible"
"Andrei Mironov" "Twelve Chairs"
"Sharon Stone" "Total Recall"
"Ryoko Hirosue" "Wasabi"
"Arnold Schwarzenegger" "Terminator"
"Jean Reno" "Wasabi"
"Peter Falk" "Colombo"
"Anatoli Papanov" "Twelve Chairs"
"Jean Reno" "Leon"
```

output.txt

```
"Andrei Mironov" 1941 "USSR" "Andrei Mironov" "Twelve Chairs"
"Arnold Schwarzenegger" 1947 "USA" "Arnold Schwarzenegger" "Total
Recall"
"Arnold Schwarzenegger" 1947 "USA" "Arnold Schwarzenegger" "Terminator"
"Jean Reno" 1948 "France" "Jean Reno" "Leon"
"Jean Reno" 1948 "France" "Jean Reno" "Mission: Impossible"
"Jean Reno" 1948 "France" "Jean Reno" "Wasabi"
"Peter Falk" 1927 "USA" "Peter Falk" "Colombo"
"Ryoko Hirosue" 1980 "Japan" "Ryoko Hirosue" "Wasabi"
"Sharon Stone" 1958 "USA" "Sharon Stone" "Basic Instinct"
"Sharon Stone" 1958 "USA" "Sharon Stone" "Total Recall"
"Tom Cruise" 1962 "USA" "Tom Cruise" "Mission: Impossible"
```

input.txt

```
5
"O]V| c -(SZ9mY ~‘/{8" 1950 "bo"
" Q G*u4 T:Eqy’d" 1979 "9"
"0" 2005 "jMsB"
" w cQ" 1982 "&"
" Q G*u4 T:Eqy’d" 2003 " J|hL"
7
" Q G*u4 T:Eqy’d" "6q*EDh!"
"0" "s"
" Q G*u4 T:Eqy’d" "CQMG::dw{"
":D h%$  W^cr" "%‘De!Si"
" Q G*u4 T:Eqy’d" "]Zo"
" Q G*u4 T:Eqy’d" "t"
"0" "Uxb/.& "
```

output.txt

```
" Q G*u4 T:Eqy’d" 1979 "9" " Q G*u4 T:Eqy’d" "6q*EDh!"
" Q G*u4 T:Eqy’d" 1979 "9" " Q G*u4 T:Eqy’d" "CQMG::dw{"
" Q G*u4 T:Eqy’d" 1979 "9" " Q G*u4 T:Eqy’d" "]Zo"
" Q G*u4 T:Eqy’d" 1979 "9" " Q G*u4 T:Eqy’d" "t"
"0" 2005 "jMsB" "0" "s"
"0" 2005 "jMsB" "0" "Uxb/.& "
" Q G*u4 T:Eqy’d" 2003 " J|hL" " Q G*u4 T:Eqy’d" "6q*EDh!"
" Q G*u4 T:Eqy’d" 2003 " J|hL" " Q G*u4 T:Eqy’d" "CQMG::dw{"
" Q G*u4 T:Eqy’d" 2003 " J|hL" " Q G*u4 T:Eqy’d" "]Zo"
" Q G*u4 T:Eqy’d" 2003 " J|hL" " Q G*u4 T:Eqy’d" "t"
```

Задача 10. Выравнивание структур

Источник:	повышенной сложности
Имя входного файла:	`input.txt`
Имя выходного файла:	`output.txt`
Ограничение по времени:	1 секунда
Ограничение по памяти:	разумное

Петя изучает язык C, и только что дошёл до структур. Его особенно заинтересовал тот факт, что поля структуры не всегда размещаются в памяти подряд: иногда между соседними полями появляются дополнительные «пустоты» (padding). Из-за этого получается, что размер структуры может зависеть от того, в каком порядке программист перечислит поля структуры! Очень интересна в этом плане, например, вот эта статья.

Дано описание структуры на языке C. В первой строке написано ключевое слово `struct` и открывающая фигурная скобка. В последней строке записана закрывающая фигурная скобка и точка с запятой. Каждая строка между ними описывает ровно одно поле структуры.

В описании поля сначала стоит тип поля, затем имя поля, и наконец точка с запятой. Тип поля отделён от имени поля хотя бы одним пробелом. Имя поля — это непустая строка, состоящая из латинских букв любого регистра, цифр и символов подчёркивания, причём имя точно **не** начинается с цифры.

В качестве типа поля может быть указан либо примитивный тип, либо указатель на примитивный тип. Во входных данных могут быть только следующие примитивные типы: `char`, `short`, `int`, `long`, `float`, `int64_t` и `double`. Указатель может быть любого порядка (т.е. двойной указатель, тройной указатель и т.п.).

В данной задаче будем считать, что структура размещается в памяти следующим образом (в реальности правила размещения определяются компилятором). Первое поле структуры помещается по адресу с максимальным выравниванием: будем считать, что его адрес делится нацело на 8. Каждое следующее поле размещается по такому адресу, что:

1. оно расположено после предыдущего,
2. оно корректно выровнено, то есть его адрес делится на его размер, и
3. его адрес минимален при выполнении первых двух условий.

При этом между соседними полями может появиться пустое место (padding) размером от 1 до 7 байт.

После последнего поля также может быть добавлено несколько байт пустоты. Это необходимо, чтобы можно было хранить массив таких структур, то есть располагать в памяти много одинаковых структур друг за другом. В конце структуры добавляется минимальное количество байт пустоты, так чтобы в массиве любой длины все поля всех структур были корректно выровнены. При этом размер всей структуры равен сумме размеров всех полей и размеров всех имеющихся в структуре пустот.

Петя может переставлять местами поля структуры. Он хочет узнать, какой при этом может получиться минимальный размер структуры, и какой максимальный размер. Как известно, размер также зависит от модели данных. Петя хочет узнать ответ для всех популярных моделей.

Формат входных данных

Во входном файле описана одна структура согласно указанным в условии правилам. В любом месте описания может быть добавлено любое количество пробелов (ASCII 32), если их добавление не разрывает на части название примитивного типа, имя поля или ключевое слово.

Гарантируется, что в структуре от 1 до 100 полей, а общее количество символов в описании не превышает 5 000. Имена всех полей структуры отличаются друг от друга и не совпадают с ключевыми словами языка C.

Формат выходных данных

В выходном файле должно быть четыре строки, по одной строке на модель данных. В каждой строке нужно записать через пробел два целых числа: минимально возможный размер структуры и максимально возможный размер.

В первой строке должен быть ответ для модели данных `LP32`, во второй — для модели `ILP32`, в третьей — для модели `LLP64` и в четвёртой — для модели `LP64`. Информацию о том, сколько байтов занимает каждый тип на каждой модели данных, можно найти на странице: `https://ru.cppreference.com/w/cpp/language/types`

Пример

input.txt	output.txt

`struct  {`
`  int x;`
`  int64_t* * Y;`
`  char   _temp1;`
`} ;`
	8 12
	12 12
	16 24
	16 24

Пояснение к примеру

Рассмотрим структуру в модели принятой на Win64 модели данных `LLP64`.

Поле `x` имеет размер 4 байта, поле `Y` размера 8 байт, а поле `_temp1` занимает 1 байт.

Если не менять порядок полей, тогда поле `x` будет занимать байты 0-3. Придётся добавить 4 байта пустоты, чтобы поле `Y` было выровнено по 8 байтам, заняв байты 8-15. Поле `_temp1` будет иметь адрес 16 и занимать один байт, но после него нужно добавить ещё 7 байтов пустоты. Без этого поле `Y` второй структуры в массиве не может быть корректно выровнено. Получается общий размер 24 байта.

Если выбирать порядок полей, то размер 24 байта получается только когда поле `Y` находится между другими двумя полями. Так получается в 2 способах среди всех 6 спопобов выбора порядка. В остальных случаях размер структуры равен 16 байтам.