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. Развернуть строку

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

Требуется реализовать функцию, которая будет разворачивать заданную строку.

Функция должна иметь сигнатуру:

```
void reverse(char *start, int len);
```

Здесь `start` — указатель на начало строки (т.е. на первый её символ), а `len` — количество символов в ней (исключая завершающий нулевой символ).

Формат входных данных

В первой строке содержится целое число N — количество строк в файле ($2 \leqslant N \leqslant 1\,000$). В каждой из следующих N строк записана одна строка, которую нужно развернуть. Строка состоит только из символов латинского алфавита, и её длина лежит в диапазоне от 1 до 100 включительно.

Формат выходных данных

Нужно вывести N развёрнутых строк.

Пример

input.txt	output.txt
4 C is cool language	C si looc egaugnal

Задача 3. Склеить строки

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

Требуется реализовать функцию, которая будет конкатенировать заданные строки. Если конкретнее, она должна дописывать вторую строку в конец первой строки.

Функция должна иметь сигнатуру:

```
char* concat(char *pref, char *suff);
```

Здесь `pref` — указатель на начало первой строки, а `suff` — указатель на начало второй строки. Нужно дописать содержимое строки `suff` в конец строки `pref`. Предполагается, что вызывающий гарантирует, что в буфере `pref` хватит места на дописываемую строку.

Обе входных строки заканчиваются нулевым символом, и склеенная строка также должна заканчиваться на него. В качестве возвращаемого значения нужно вернуть указатель на конец (т.е. на нулевой символ) склеенной строки.

В качестве тестовой задачи нужно объединить все заданные строки.

Формат входных данных

В первой строке содержится целое число N — количество строк в файле ($2 \leqslant N \leqslant 10\,000$). В каждой из следующих N строк записана одна строка. Строка состоит только из символов латинского алфавита, и её длина лежит в диапазоне от 1 до 100 включительно.

Формат выходных данных

Нужно вывести одну строку, в которой объединены по порядку все N строк из входного файла.

Пример

input.txt	output.txt
4 C is cool language	Ciscoollanguage

Задача 4. Имена и возрасты

Источник:	основная
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 5. Сколько буков

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

Требуется реализовать функцию, которая будет определять, сколько в строке больших букв, маленьких букв и цифр.

Функция должна иметь сигнатуру:

```
int calcLetters(char *iStr, int *oLowerCnt, int *oUpperCnt, int *oDigitsCnt);
```

Здесь `iStr` — указатель на начало строки, завершающейся нулевым символом. Параметры `oLowerCnt`, `oUpperCnt` и `oDigitsCnt` выходные: вызывающий передаёт в них указатель на какие-нибудь локальные переменные, чтобы получить в них соответствующий результат. Функция возвращает длину строки `iStr`, в переменную `*oLowerCnt` нужно записать количество маленьких букв, в `*oUpperCnt` записать количество больших букв, а в `*oDigitsCnt` записать количество цифр.

В качестве тестовой задачи нужно прочитать все строки файла и распечатать статистику для каждой из них.

Формат входных данных

Строки файла могут содержать любые печатаемые символы ASCII, включая пробелы (коды от 32 до 126 включительно). Поэтому рекомендуется использовать `gets` для чтения строк.

Длина любой строки не превышает 100, строки могут быть пустыми. Учтите, что последняя строка файла также завершается символом перевода строки.

Формат выходных данных

Для каждой строки входного файла выведите статистику ровно в том же формате, как в примере выходных данных.

Пример

input.txt

```
/*C-style comments can contain
multiple lines*/ /*or just one*/
// C++-style comment lines
int main() {
  // The below code wont be run
  //return 1;
  return 0; //this will be run
}
```

output.txt

```
Line 1 has 30 chars: 24 are letters (23 lower, 1 upper), 0 are digits.
Line 2 has 32 chars: 22 are letters (22 lower, 0 upper), 0 are digits.
Line 3 has 26 chars: 18 are letters (17 lower, 1 upper), 0 are digits.
Line 4 has 12 chars: 7 are letters (7 lower, 0 upper), 0 are digits.
Line 5 has 31 chars: 21 are letters (20 lower, 1 upper), 0 are digits.
Line 6 has 13 chars: 6 are letters (6 lower, 0 upper), 1 are digits.
Line 7 has 30 chars: 19 are letters (19 lower, 0 upper), 1 are digits.
Line 8 has 1 chars: 0 are letters (0 lower, 0 upper), 0 are digits.
```

Задача 6. Прочитать время

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

Требуется реализовать функцию, будет считывать время из заданной строки.

Функция должна иметь сигнатуру:

```
int readTime(char *iStr, int *oHours, int *oMinutes, int *oSeconds);
```

Здесь `iStr` — указатель на строку, в которой должно быть записано время. Параметры `oHours`, `oMinutes`, `oSeconds` — выходные параметры, т.е. вызывающий должен передать в них указатель на свои локальные переменные, куда будет записаны соответсвующие результаты: `*oHours` — количество часов, `oMinutes` — количество минут, `oSeconds` — количество секунд. Функция должна возвращать код возврата: `1`, если прочитать время удалось, и `0` в случае неудачи.

Параметры `oMinutes`, `oSeconds` опциональные: вызывающий может передать в них NULL, если его, например, не интересует количество секунд. При этом если указатель `oMinutes` нулевой, то и указатель `oSeconds` тоже должен быть нулевой.

Время записано в формате *H:M:S* или *H:M*. То есть строка состоит из двух или трёх частей, отделённых друг от друга одним двоеточием. Каждая часть — это целое число из одной или двух цифр, возможно с ведущими нулями. При этом если задано две части, то это часы и минуты, а если три — то это часы, минуты и секунды. Заметим, что часы должны быть в диапазоне до 0 до 23, а минуты и секунды в диапазоне от 0 до 59.

Используя функцию, нужно решить тестовую задачу. В файле записана тестовая строка длины от 3 до 15 символов без пробелов. Нужно применить функцию `readTime` к каждой строке и распечатать результат её вызова.

Нужно вызвать функцию три раза:

1. Указатели `oHours`, `oMinutes`, `oSeconds` ненулевые. После вызова нужно распечатать код возврата и числа `*oHours`, `*oMinutes`, `*oSeconds` через пробел.
2. Указатель `oSeconds` нулевой. После вызова нужно распечатать код возврата и числа `*oHours`, `*oMinutes` через пробел.
3. Указатели `oMinutes` и `oSeconds` нулевые. После вызова нужно распечатать код возврата и число `*oHours` через пробел.

Заметим, что вызывать функцию три раза подряд по сути бессмысленно: мы делаем это только для того, чтобы протестировать все случаи с нулевыми указателями.

Формат входных данных

В единственной строке задано время в формате, указанном выше (с секундами или без).

Время на входе также может быть указано неверно. Чтобы не было разногласий, в каком случае считать время корректным, а в каком нет, гарантируется, что во всех тестах будут только ошибки двух видов:

1. Формат полностью соответствует условию, но число часов, минут или секунд выходит за пределы допустимого диапазона.
2. Формат некорректный: взято корректно записанное время, и между каждой парой символов в строке вставлен символ вертикальной черты '`|`' (ASCII 124).

Таким образом, вы можете самостоятельно решить, считать ли случаи вроде `12:001:35` или `17:0xy` корректными или нет: в тестах таких ситуаций не будет.

Формат выходных данных

Нужно вывести три строки с 4-мя, 3-мя и 2-мя целыми числами соответственно (см. описание выше в условии).

Заметьте, что если дата задана неверно, то все числа кроме кода возврата должны быть равны -1.

Пример

input.txt	output.txt
15:01:13	1 15 1 13 1 15 1 1 15
7:9	1 7 9 0 1 7 9 1 7
7:99	0 -1 -1 -1 0 -1 -1 0 -1
1|3|:|5|:|1|0	0 -1 -1 -1 0 -1 -1 0 -1

Комментарий

Вам может пригодиться функция `sscanf` и её возвращаемое значение.

Пояснение к примеру

В первом примере указаны секунды, а во втором — нет. В третьем тесте 99 минут, что выходит за пределы диапазона, поэтому возвращается неуспех. Четвертый тест неверно отформатирован: это время `13:5:10` со вставленными вертикальными чертами.

Задача 7. 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 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 8. Числа Фибоначчи

Источник:	повышенной сложности
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 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.