Задача 1. Распечатать время

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 распечатывает в выходной файл время, и решить с её помощью тестовую задачу.

Функция должна иметь сигнатуру:

```
void printTime(int h, int m, int s);
```

Здесь `h` — количество часов (от 0 до 23), а `m` и `h` — количество минут и секунд соответственно (от 0 до 59).

Формат входных данных

В первой строке содержится целое число N — количество времён в файле ($2 \leqslant N \leqslant 1\,000$).
В каждой из следующих N строк описано время в формате трёх целых чисел: `h`, `m`, `s`.

Формат выходных данных

Нужно вывести N строк, по одной дате в строке, в формате `HH:MM:SS` (см. пример).

Пример

input.txt	output.txt
3 12 37 59 1 1 1 23 59 59	12:37:59 01:01:01 23:59:59

Комментарий

Вам может пригодиться формат `"%02d"`: он печатает целое число, дополняя его слева нулями до двух символов.

Задача 2. Прочитать время

Источник:	базовая*
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 3. 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 4. 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 5. Сколько буков

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 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. Развернуть строку

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 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 7. Разделить на слова

Источник:	основная*
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 8. Склеить строки

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 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 9. Хранение строк

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

Нужно реализовать систему хранения строк в динамической памяти.

Система должна обрабатывать три вида запросов/операций:

0. *Создать строку*: указывается длина создаваемой строки и собственно содержимое строки. Нужно выделить блок памяти при помощи `malloc` и записать в него содержимое.
1. *Удалить строку*: указывается идентификатор ранее созданной строки. Нужно освободить память, выделенную ранее для этой строки, при помощи `free`.
2. *Вывести строку*: указывается идентификатор ранее созданной строки. Нужно распечатать содержимое этой строки в выходной файл.
3. *Сколько символов в строке*: указывается идентификатор ранее созданной строки и один символ. Нужно вывести в выходной файл одно целое число: сколько раз этот символ встречается в указанной строке.

Идентификатором строки является номер запроса на её создание в общей нумерации запросов. При обращении к строке по идентификатору гарантируется, что эта строка ещё **не** была удалена.

Замечание: Не все созданные строки удаляются в запросах. В целях аккуратности удалите все остающиеся строки при помощи `free` в конце программы.

Формат входных данных

В первой строке записано целое число N — количество количество запросов ($1 \leqslant N \leqslant 10^5$). В остальных N строках описаны запросы, по одному запросу в строке.

Запрос начинается с целого числа t, обозначающего тип запроса. Если $t = 0$, то это запрос создания, и тогда далее указана длина строки l и сама строка ($1 \leqslant l \leqslant 10^5$). Если $t = 1$, то это запрос удаления, а если $t = 2$ — то это запрос на вывод. В обоих случаях далее указано целое число k — идентификатор строки ($0 \leqslant k < N$). Если $t = 3$, то это запрос о количестве символов, и тогда далее указан идентификатор строки k и ещё один символ через пробел.

Все строки состоят из произвольных печатаемых символов ASCII (коды от 33 до 126 включительно). То же верно и про символы в запросах на количество.

Сумма всех длин l по всем запросам создания не превышает $5 \cdot 10^5$. Суммарный размер всех строк, которые вам нужно вывести, не превышает $5 \cdot 10^5$. Сумма длин строк по всем запросам о количестве символов не превышает 10^8.

Пример

input.txt	output.txt
12 0 3 aba 2 0 3 0 a 3 0 b 1 0 0 6 malloc 0 5 %d%s% 2 1 2 2 1 1 3 2 % 2 2	aba 2 1 malloc %d%s% 3 %d%s%

Пояснение к примеру

Сначала создаётся строка “aba” с идентификатором 0. Далее она распечатывается, и выводится количество букв ’a’ и ’b’ в ней. Наконец, нулевая строка удаляется.

Затем создаются ещё две строки: строка “malloc” и строка “%d%s%” с идентификаторами 1 и 2 соответственно. Потом они распечатываются и строка “malloc” удаляется. В конце выводится количество процентов в строке “%d%s%” и она распечатывается ещё раз.

Задача 10. 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`.