Задача 1. Библиотека линейной алгебры

Источник:	базовая I
Имя входного файла:	--
Имя выходного файла:	--
Ограничение по времени:	разумное
Ограничение по памяти:	разумное

Код задачи: `3/1_blas`

Для выполнения вычислений над векторами и матрицами большого размера обычно используют библиотеку BLAS (Basic Linear Algebra Subprograms). Данная библиотека содержит несколько десятков функций, расфасованных по трём уровням согласно асимптотическому времени работы. Более сложные операции линейной алгебры (вроде решения систем уравнения) обычно реализуют с использованием функций BLAS. Существует множество реализаций библиотеки BLAS: проприетарные и open-source, простые и оптимизированные, для обычного процессора, видеокарты или суперкомпьютера, и так далее. Конечно, большая часть реализаций оптимизируется до предела, чтобы выжать максимальную производительность из конкретной аппаратуры.

В этой задаче предлагается написать простую реализацию некоторых функций BLAS. Сигнатуры функций настоящего BLAS очень громоздкие (например, функция перемножения матриц принимает 13 параметров), поэтому для данной задачи они несколько упрощены. Будем считать, что матрица размера m на n хранится в памяти в row-major формате без пропусков, то есть если указатель на начало матрицы равен `pMatr`, то элемент матрицы в i-ой строке в j-ом столбце лежит в `pMatr[i * n + j]`. Все векторы, матрицы и скаляры в данной задаче содержат вещественные числа типа `double`.

Библиотека должна реализовывать все функции из представленного на следующей странице хедера `myblas.h`.

На проверку требуется отправить три файла: `level1.c`, `level2.c` и `level3.c`. В этих файлах должны быть реализованы функции первого, второго и третьего уровня соответственно. Хедер `myblas.h` с указанным выше содержимым будет доступен в текущей директории.

Кроме того, вы должны научиться собирать из вашего кода статическую библиотеку. Для этого нужно дополнительно отправить два файла:

- `c.bat.txt` — скрипт для сборки `myblas.lib` компилятором MSVC.
- `c.sh.txt` — скрипт для сборки `myblas.a` компилятором GCC.

Каждый скрипт сборки должен содержать одну или несколько команд, которые нужно запускать в командной строке. Команды будут запускаться по очереди в той же директории, куда будет распакованы/сложены все ваши файлы. Разрешается вызывать программы: `cl`, `lib`, `gcc`, `ar`, путь к ним указывать **не** нужно. Заметьте, что если переименовать `c.bat.txt` или `c.sh.txt` в `c.bat`, то его можно будет запустить локально.

Тестирующий код жюри будет корректно вызывать эти функции, все размеры будут положительными и не будут превышать 1 000. Кроме того, гарантируется, что при вызове любой функции все параметры-указатели будет указывать на неперекрывающиеся куски памяти. Общее количество вызовов и размеры будут достаточно малы, чтобы решение с правильной асимптотикой легко укладывалось по времени. Для вашего удоства при обнаружении ошибки в лог будет писаться причина ошибки: имя функции, и иногда даже входные данные.

Содержимое хедера `myblas.h`:

```
#pragma once

//====== уровень 1 ======
// (все векторы длины n)

//скопировать вектор из X в Y
void dcopy(int n, const double *X, double *Y);
//обменять местами содержимое векторов X и Y
void dswap(int n, double *X, double *Y);
//домножить вектор X на коэффициент alpha
void dscal(int n, double alpha, double *X);
//прибавить к вектору Y вектор X, умноженный на коэффициент alpha
void daxpy(int n, double alpha, const double *X, double *Y);
//вычислить скалярное произведение векторов X и Y
double ddot(int n, const double *X, const double *Y);

//====== уровень 2 ======

//вычислить вектор (alpha*A*X + beta*Y) длины m, и записать его в Y
//здесь A -- матрица размера m на n, X -- вектор длины n, а Y -- вектор длины m
void dgemv(
    int m, int n,
    double alpha, const double *A, const double *X,
    double beta, double *Y
);
//вычислить матрицу (alpha*X*Yt + A) и записать её в A
//здесь Yt -- это транспонированный вектор Y, то есть записанный как вектор-строка
// A -- матрица размера m на n, X -- вектор длины m, а Y -- вектор длины n
void dger(
    int m, int n,
    double alpha, const double *X, const double *Y,
    double *A
);

//====== уровень 3 ======

//вычислить матрицу (alpha*A*B + beta*C) и записать её в C
//здесь A -- матрица размера m на k, B -- матрица размера k на n,
// C -- матрица размера m на n
void dgemm(
    int m, int n, int k,
    double alpha, const double *A, const double *B,
    double beta, double *C
);
```

Задача 2. Сборка

Источник:	базовая I
Имя входного файла:	`--`
Имя выходного файла:	`--`
Ограничение по времени:	—
Ограничение по памяти:	—

Код задачи: `3/2_build`

В этой задаче предлагается написать простой скрипт для сборки заданной программы. Программа использует библиотеки zlib и minizip, чтобы прочитать текстовый файл из zip-архива, а затем выводит гистограмму символов этого текста. Вам нужно собрать и саму программу, и эти библиотеки. Полный исходный код программы можно скачать по этой ссылке.

При сборке следует иметь ввиду следующее:

- Из zlib нужно линковать все файлы исходного кода, а из minizip — только `unzip.c` и `ioapi.c`.
- В файлах подключаются хедеры без правильного пути — вам придётся настроить include paths.
- Входной zip-файл запаролен. Чтобы включить в minizip поддержку паролей, нужно определить макрос `USE_CRYPT`.

На проверку требуется отправить один файл: `c.bat.txt`. Это скрипт сборки, который должен содержать одну или несколько команд, которые будут запускаться в командной строке. Команды будут запускаться по очереди в той же директории, куда будет распаковано содержимое вышеуказанного архива с исходным кодом (файл `test.c` будет в той же директории). В результате сборки должен получиться файл `checked_solution.exe` в той же директории.

Разрешается вызывать программы: `cl`, `lib`, `gcc`, `ar`, путь к ним указывать **не** нужно. Среди этих небогатых возможностей вы можете самостоятельно выбрать способ сборки и компилятор. Учтите, что у вас нет возможности копировать файлы, удалять файлы, и менять текущую директорию — придётся думать, как обойтись без этого. Заметьте, что если переименовать `c.bat.txt` в `c.bat`, то его можно будет запустить локально.

Задача 3. Арифметические выражения

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

В данной задаче предлагается вычислить заданное арифметическое выражение. Выражение состоит из целочисленных констант, над которыми выполняются операции сложения, вычитания, умножения и деления. Части выражения могут быть заключены в скобки. Унарный минус разрешается, если непосредственно до него нет знака другой операции. Между токенами может быть любое количество пробелов, а может и не быть пробелов.

Вычислять выражение нужно в вещественных числах. Гарантируется, что не будет деления на ноль и проблем с точностью, и что все целочисленные константы длиной не больше четырёх цифр.

Формат входных данных

В единственной строке дано выражение, которое нужно вычислить. Гарантируется, что выражение корректно. Длина выражения не превышает 500 000 символов.

Формат выходных данных

Нужно вывести одно вещественное число — значение выражения. Рекомендуется выводить число с максимальной точностью.

Абсолютная или относительная ошибка вашего ответа **не** должна превышать 10^{-12}.

Пример

input.txt
`-5+  4 *7- 3* 2  +  17/3- 5 + (3 - 5 + 3) * 2`
output.txt
`19.66666666666666785090`

input.txt
`((((3)) + ((-5))))`
output.txt
`-2.00000000000000000000`

Комментарий

При рекурсивном разборе выражения с большой вложенностью скобок нужен довольно большой программный стек — явно больше, чем 1 мегабайт по умолчанию. При решении на Visual C, настоятельно рекомендуется увеличить стек, добавив следующую строчку в самом начале программы:

```
#pragma comment(linker, "/STACK:50000000")
```

Задача 4. Длина кривой

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

В компьютерной графике и геометрическом моделировании активно используются NURBS-кривые. Чтобы уметь с ними работать, надо хорошо понимать, как работают полиномиальные сплайны. В этой задаче мы будем иметь дело с кубическими сплайновыми кривыми — это лишь простый частный случай из класса NURBS-кривых. От вас требуется найти длину заданной кривой.

Параметрическая кривая в 3D-пространстве определяется тремя координатными функциями $x(t)$, $y(t)$ и $z(t)$. Здесь t — это какой-то абстрактный параметр (например время), а значения функций x, y, z определяют координаты точки в 3D-пространстве. У кривой есть также интервал определения $[a, b]$. Если плавно увеличивать значение t от $t = a$ (начало) до $t = b$ (конец), то при этом точка с координатами $x(t)$, $y(t)$ и $z(t)$ пройдёт по всей заданной кривой, без остановок и разворотов.

В этой задаче вам дана кубическая сплайновая кривая. Это означает, что интервал её определения разбивается на N подынтервалов, которые называются *спанами*. Если рассматривать кривую на любом отдельном спане, то координатные функции $x(t)$, $y(t)$ и $z(t)$ являются кубическими многочленами от t. При этом на каждом спане эти кубические многочлены свои, и между спанами могут не совпадать. Гарантируется, что многочлены на соседних спанах согласованы, так что кривая непрерывная — без скачков и разрывов.

Нужно найти длину заданной кривой в 3D пространстве.

Формат входных данных

В первой строке дано одно целое число N — количество спанов ($1 \leqslant N \leqslant 1\,000$). Далее идут описания N спанов в порядке увеличения параметра t.

Описание каждого спана начинается с двух чисел l и r, которые определяют подынтервал. То есть приведённые далее многочлены действуют лишь при значениях параметра $t \in [l, r]$. Разумеется, выполняется $l < r$ и число l совпадает с числом r предыдущего спана.

Далее в описании спана задаются функции $x(t)$, $y(t)$ и $z(t)$, каждая функция задана многочленом в отдельной строке. Каждый многочлен $p(t)$ задан четырьмя числами c_0, c_1, c_2, c_3 и определяется так:

$$p(t) = c_0 + c_1(t - l) + c_2(t - l)^2 + c_3(t - l)^3$$

Обратите внимание, что параметр t сдвигается на l влево перед тем, как вычисляется кубический многочлен.

Все числа в описании спана вещественные. Числа будут лежать в разумных пределах, спаны будут иметь разумную длину, и в целом кривая будет хорошо себя вести =) Гарантируется, что соседние спаны непрерывно соединяются, то есть дыр между спанами нет.

Формат выходных данных

Нужно вывести одно вещественное число — длину заданной кривой. Рекомендуется выводить число с максимальной точностью.

Абсолютная или относительная ошибка вашего ответа **не** должна превышать 10^{-8}.

Пример

input.txt	output.txt
3 0 1 1 0 0 0 0 1 0 0 5 0 0 0 1 1.5 1 0 6 -4 1 3 0 -4 5 0 0 0 1.5 4 2 0.4 0 0 2 0 0 0 5 0 0 0	3.54886798803189495999

Пояснение к примеру

На отрезке $t \in [0, 1]$ кривая определяется функциями:

$$x(t) = 1;\ y(t) = t;\ z(t) = 5$$

На отрезке $t \in [1, \frac{3}{2}]$ действуют функции:

$$\begin{aligned} x(t) &= 1 + 6(t-1)^2 - 4(t-1)^3; \\ y(t) &= 1 + 3(t-1) - 4(t-1)^3; \\ z(t) &= 5 \end{aligned}$$

На отрезке $t \in [\frac{3}{2}, 4]$ координаты изменяются так:

$$x(t) = 2 + \frac{4}{10}\left(t - \frac{3}{2}\right);\ y(t) = 2;\ z(t) = 5$$

Комментарий

Чтобы решить задачу, нужно заметить, что длина кривой вычисляется как интеграл:

$$L = \int_{t=a}^{b} \sqrt{\dot{x}^2(t) + \dot{y}^2(t) + \dot{z}^2(t)}dt$$

Здесь $\dot{x}(t)$, $\dot{y}(t)$, $\dot{z}(t)$ — это производные координатных функций.

Вместо того, чтобы пытаться проинтегрировать это выражение математически, лучше вычислить интеграл численно. Для этого рекомендуется использовать правило Симпсона — оно даёт хорошую точность. Чтобы его применить, достаточно научиться вычислять подынтегральное выражение при любом значении t.

Учтите, что правило Симпсона даёт хорошую точность только на гладких функциях, а на стыке спанов производные кривой могут иметь скачок. По этой причине следует вычислить интеграл по правилу Симпсона отдельно на каждом спане, а потом сложить результаты.

Задача 5. Снова растущий массив

Источник:	основная I
Имя входного файла:	---
Имя выходного файла:	---
Ограничение по времени:	5 секунд*
Ограничение по памяти:	разумное

Код задачи: `3/5_vector`

В этой задаче предлагается реализовать псевдошаблонный код растущего массива, работающий для элементов любого типа. Этот код в дальнейшем можно легко подключить в любой задаче.

Вам нужно отправить два файла: `array_decl.h` и `array_def.h`. Файл `array_decl.h` будет многократно подключаться в хедере, чтобы сгенерировать объявление структуры и всех необходимых функций для работы с ней. Файл `array_def.h` будет многократно подключаться в файле исходного кода, чтобы сгенерировать определения всех функций, объявленных с помощью `array_decl.h`. Разрешается также отправить дополнительные хедеры.

Непосредственно перед каждым включением любого из ваших хедеров будет устанавлено два макроса `TYPE` и `NAME`. Первый задаёт известный на момент компиляции тип элемента, а второй определяет название массива (гарантируется, что он является корректным именем языка C). Например, чтобы объявить или определить массив под названием `array_long` с элементами типа `long`, перед включением будут установлены макросы:

```
#define TYPE long
#define NAME array_long
```

Разумеется, включения `array_decl.h` в хедере будут точно соответствовать включениям `array_decl.c` в файле с исходным кодом по количеству и типам и именам.

Растущий массив должен автоматически увеличивать размер буфера памяти по мере необходимости, так чтобы метод `push` работал за амортизированное время $O(1)$. В данной задаче запрещается автоматически уменьшать размер буфера памяти. Любой метод кроме `destroy` может лишь увеличить размер буфера, но не должен уменьшать его.

Ваши файлы должны объявлять/определять следующие вещи:

```
typedef struct array_long {
    int n;          //number of elements in array
    long *arr;      //points to the array of elements
    ...             //any other members can be added here
} array_long;

//initializes members of [vec] structure for empty array
void array_long_init(array_long *vec);

//makes array [vec] empty and frees its array buffer [vec->arr]
//note: this function can be called many times
void array_long_destroy(array_long *vec);

//adds element [value] to the end of array [vec]
//returns index of the added element
int array_long_push(array_long *vec, long value);

//removes the last element from array [vec]
//returns removed element
long array_long_pop(array_long *vec);

//ensures that array [vec] has enough storage for [capacity] elements
//note: address of elements surely won't change before [vec->n] exceeds capacity
void array_long_reserve(array_long *vec, int capacity);

//changes number of elements in array [vec] to [newCnt]
//if the number increases, new elements get value [fill]
//if the number decreases, some elements at the end are removed
void array_long_resize(array_long *vec, int newCnt, long fill);

//inserts elements [arr[0]], [arr[1]], [arr[2]], ..., [arr[num-1]]
//in-between elements [vec->arr[where-1]] and [vec->arr[where]]
//note: the whole array [arr] cannot be part of array [vec]
//[where] may vary from 0 to [vec->n], [num] can also be zero
void array_long_insert(array_long *vec, int where, long *arr, int num);

//removes elements [vec->arr[k]] for k = [where], [where+1], ..., [where+num-1]
//all the elements starting from [where+num]-th are shifted left by [num] positions
void array_long_erase(array_long *vec, int where, int num);
```

Указанные выше объявления проименованы для случая, когда `TYPE` имеет значение `long`, а имя `NAME` равно `array_long`. Если макросы имеют другое значение, имена структуры и функций также будут другими. Например, можно сделать массив указателей так:

```
#define TYPE void *
#define NAME array_pvoid
```

Тогда функция добавления элемента будет иметь сигнатуру:

```
int array_pvoid_push(array_pvoid *vec, void *value);
```

При тестировании ваш код будет включаться в тестовый код жюри согласно правилам. Ниже приведён пример того, как примерно будут использоваться ваши хедеры.

Хедер `sample.h`:

```
#pragma once

#define TYPE double
#define NAME vector
#include "array_decl.h"

#define TYPE int
#define NAME indices
#include "array_decl.h"
```

Файл исходного кода sample.c:

```c
#include "sample.h"
#include <assert.h>

#define TYPE double
#define NAME vector
#include "array_def.h"

#define TYPE int
#define NAME indices
#include "array_def.h"

int main() {
    vector values;
    vector_init(&values);
    assert(values.n == 0);
    vector_push(&values, 1.0);
    vector_push(&values, 2.0);
    assert(values.n == 2 && values.arr[0] == 1.0 && values.arr[1] == 2.0);
    indices ids;
    indices_init(&ids);
    int temp[] = {1,2,3,4,5,6};
    indices_insert(&ids, 0, temp, sizeof(temp)/sizeof(temp[0]));
    assert(ids.n == 6 && ids.arr[3] == 4);
    indices_insert(&ids, 4, temp, 5);
    assert(ids.n == 11 && ids.arr[3] == 4 && ids.arr[9] == 5);
    assert(ids.arr[4] == 1 && ids.arr[6] == 3 && ids.arr[8] == 5);
    indices_erase(&ids, 2, 5);
    assert(ids.n == 6 && ids.arr[0] == 1 && ids.arr[1] == 2);
    assert(ids.arr[2] == 4 && ids.arr[3] == 5 && ids.arr[4] == 5);
    indices_destroy(&ids); //memory freed
    indices_push(&ids, 13);
    assert(ids.n == 1 && ids.arr[0] == 13);
    indices_resize(&ids, 5, -1);
    assert(ids.n == 5 && ids.arr[0] == 13 && ids.arr[1] == -1);
    indices_reserve(&ids, 1000);
    int *ptr = &ids.arr[0];
    for (int i = 0; i < 900; i++) indices_push(&ids, i);
    assert(*ptr == 13 && ptr == &ids.arr[0]);
    for (int i = 0; i < 900; i++) indices_pop(&ids);
    assert(ptr == &ids.arr[0]); //never shrink buffer!
    indices_destroy(&ids);
}
```

Задача 6. Чтение JSON

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

Код задачи: `3/6_json`

В этой задаче нужно прочитать JSON с помощью библиотеки json-c и записать его в формате BSON.

Спецификация формата JSON есть на сайте json.org. Любое *значение* в формате JSON имеет один из следующих типов:

- Строка — произвольная последовательность символов Unicode, задаётся в кодировке UTF-8 и заключается в двойные кавычки
- Число — в общем случае вещественное с двойной точностью
- Булево значение `true` или `false`
- Нулевое значение `null`
- Массив — элементы могут быть любыми значениями: они разделяются запятыми, всё содержимое массива заключается в квадратные скобки
- Объект — список пар вида «ключ: значение», где ключ — это строка, а значение может быть любым значением

Во входном файле записано ровно одно значение, которое является либо объектом, либо массивом. Формат записи таков, что библиотека `json-c` нормально его читает с настройками по умолчанию.

Формат BSON является бинарным аналогом формата JSON, его спецификация есть на сайте bsonspec.org. Она может быть проще для понимания, чем идущее далее описание. В данной задаче никакая строка не будет содержать нулевой байт — в этом случае любое JSON-значение можно записать в формате BSON как последовательность байтов.

При конвертации в BSON каждое поле объекта и каждый элемент массива записывается в следующем виде: сначала один байт, кодирующий тип значения, потом имя поля, и наконец его значение. У элемента массива имя поля определяется как его номер в массиве, записанный в десятичной системе исчисления.

Следует использовать следующие типы BSON:

- Строка — тип 2
- Число — тип 1
- Булево значение — тип 8
- Нулевое значение `null` — тип 10
- Массив — тип 4
- Объект — тип 3

Значение типа «строка» записывается так: сначала 32-битное целое число, равное количеству байт в строке плюс один, затем сама строка как последовательность байтов в кодировке UTF-8, и наконец нулевой байт. Значение типа «число» записывается в 8 байт как значение типа `double`. Булево значение записывается одним байтом: 0 для `false` и 1 для `true`. Значение `null` записывается пустой последовательностью байтов. Массив или объект записывается так: сначала 32-битное целое, равное количеству байтов в записи объекта/массива, затем все элементы массива или поля массива в нужном порядке, и наконец нулевой байт.

Имя поля записывается почти как строка, но длина в начале не записывается. То есть пишется сразу содержимое строки, и в конце нулевой байт.

Что рекомендуется посмотреть по библиотеке `json-c` в первую очередь:

1. Пример парсинга test2.c.
2. Хедер json_visit.h.
3. Хедер json_object.h.

В систему тестирования нужно посылать один файл с исходным кодом, в котором можно подключать хедеры библиотеки `json-c` (без путей). При сборке ваш файл будет компилироваться и линковаться со статической библиотекой `json_c.lib`.

Пример

input.txt

```
{
  "array": [
    1,
    2,
    3
  ],
  "boolean": true,
  "color": "#82b92c",
  "null": null,
  "number": 123,
  "object": {
    "a": "b",
    "c": "d",
    "e": [ 5, {"key": -9} ]
  },
  "my string": "Hello World",
  "codename": "3/6_json"
}
```

output.bin

```
DE 00 00 00 04 61 72 72 61 79 00 26 00 00 00 01 30 00 00 00 00 00 00 00
F0 3F 01 31 00 00 00 00 00 00 00 00 40 01 32 00 00 00 00 00 00 00 08 40
00 08 62 6F 6F 6C 65 61 6E 00 01 02 63 6F 6C 6F 72 00 08 00 00 00 23 38
32 62 39 32 63 00 0A 6E 75 6C 6C 00 01 6E 75 6D 62 65 72 00 00 00 00 00
00 C0 5E 40 03 6F 62 6A 65 63 74 00 3F 00 00 00 02 61 00 02 00 00 00 62
00 02 63 00 02 00 00 00 64 00 04 65 00 25 00 00 00 01 30 00 00 00 00 00
00 00 14 40 03 31 00 12 00 00 00 01 6B 65 79 00 00 00 00 00 00 00 22 C0
00 00 00 02 6D 79 20 73 74 72 69 6E 67 00 0C 00 00 00 48 65 6C 6C 6F 20
57 6F 72 6C 64 00 02 63 6F 64 65 6E 61 6D 65 00 09 00 00 00 33 2F 36 5F
6A 73 6F 6E 00 00
```

Комментарий

Пример в нормальном виде можно скачать отсюда.

Хедеры `config.h` и `json_config.h` можно скачать отсюда. С ними можно собрать статическую библиотеку `json-c` напрямую компилятором. Без них придётся запускать CMake. В системе тестирования эти хедеры будут присутствовать, как и все другие хедеры библиотеки.

Задача 7. Чтение JSON 2

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

В данной задаче нужно самостоятельно реализовать чтение JSON-формата. Чтобы упростить задачу, все тесты соответствуют следующим ограничениям:

- JSON корректен, полностью соответствует спецификации.
- В качестве чисел (number) встречаются только целые числа (без точки, экспоненциального вида и прочей гадости). Отрицательные числа могут быть.
- Внутри каждого строкового значения содержатся только латинские буквы, цифры, знаки препинания и пробелы. Отсутствуют: обратный слэш и контрольные последовательности, всякого рода кавычки, контрольные символы (т.е. ASCII < 32), перевод строки, Unicode-символы вне ASCII.

Чтобы проверить, что вы верно разобрали JSON, вашей программе нужно также обработать запросы. Каждый запрос задаёт путь от корневого объекта или массива JSON-а до искомого значения.

Напоминаем, что структурно JSON состоит из массивов и объектов. Начинать поиск по пути нужно из корня JSON-а. Когда в пути написана строка в двойных кавычках, нужно найти в текущем объекте поле с этим именем и перейти в него. Когда в пути написан индекс в квадратных скобках, нужно найти в текущем массиве элемент с этим индексом и перейти к нему. После выполнения всех переходов поиск останавливается на том значении, которое надо вывести.

Формат входных данных

В первых N строках задаётся сам JSON. Далее идёт отдельная строка, которая содержит в точности строку: `<<<>>>`

После этого идут запросы. Каждый запрос начинается с целого числа K — сколько переходов нужно сделать, чтобы добраться от корня до искомого значения. Далее идёт K строк, в каждой из них прописан один переход: либо строка (имя поля), либо номер в квадратных скобках (индекс в массиве). После всех запросов записано одно число `-1`.

Гарантируется, что размер не превышает 1 мегабайт. Запросы будут составлены таким образом, что если вы будете искать в объекте поле с указанным именем перебором всех полей, то это будет работать достаточно быстро.

Формат выходных данных

Для каждого запроса нужно вывести искомое значение в отдельной строке. Если значение является объектом, нужно вывести `<object>`, а если массивом, то нужно вывести `<array>`. В противном случае нужно вывести само значение в точности так, как оно записано в JSON-е.

Пример

input.txt	output.txt

{
  "array": [
  1,
  2,
  3
  ],
  "boolean": true,
  "color": "#82b92c",
  "null": null,
  "number": 123,
  "object": {
  "a": "b",
  "c": "d",
  "e": [5, {"key": -9}]
  },
  "my string": "Hello World"
}
<<<>>>
4
"object"
"e"
[1]
"key"
1
"null"
1
"my string"
1
"color"
2
"array"
[2]
1
"array"
3
"object"
"e"
[1]
-1
	-9
	null
	"Hello World"
	"#82b92c"
	3
	<array>
	<object>

Комментарий

В данной задаче от вас не требуется извлекать числа, отличать числа от булевых значений, null-ов и строк. Достаточно записать соответствующие значения как строки при парсинге, и потом вывести какие-то из этих строк при запросах.

Задача 8. Чтение JSON +

Источник:	космической сложности II
Имя входного файла:	`input.txt`
Имя выходного файла:	`output.bin`
Ограничение по времени:	разумное
Ограничение по памяти:	разумное

В этой задаче нужно прочитать JSON и записать его в формате BSON. Решение должно работать так же, как в задаче «Чтение JSON», но без использования библиотеки `json-c`. Вы можете использовать решение из той задачи как проверочное при тестировании.

В тестах будут встречаться всевозможные странности, разрешённые в JSON:

1. В строках могут встречаться контрольные символы, записанные в виде `\t`, `\n`, и т.п.
2. В строках могут встречаться символы двойных кавычек и обратного слеша: `\\` и `\"`
3. В строках могут встречаться unicode-символы, записанные в виде `\u5F0a`
4. В строках могут встречаться unicode-символы, записанные в кодировке UTF-8

Эти случаи разрешены спецификацией (лучше её почитать), и `json-c` с ними справляется.

В тексте никогда и ни в каком виде не будет встречаться нулевой байт. Хоть нулевые байты и разрешены спецификацией, в данной задаче они запрещены (и в формате BSON с ними проблема).

Кроме того, в этой задаче могут встречаться вещественные числа, в том числе записанные в экспоненциальной форме, например: `1.345e-7`

Ещё в тестах иногда ставится запятая после последнего элемента в массиве или после последнего поля в объекте. Спецификация запрещает такую вольность, однако многое реализации работают с такими данными.

Наконец, в тестах будут встречаться строчные и блочные комментарии с тем же синтаксисом, как в языке C. Спецификация JSON этого не разрешает, однако `json-c` с этим отлично справляется.

Тесты проверяют описанные выше случаи примерно в порядке перечисления в тексте. Если вы проверили все эти случаи, но решение всё равно не проходит тесты, пишите. Будем смотреть =)