Занятие №5. Однопроходные алгоритмы

Часто бывает нужно обработать какую-нибудь длинную последовательность, но сохранять ее нет возможности или просто нежелательно. В самом деле, не будем же мы покупать телефонный справочник, если нам надо найти всего один телефон! Быстро просмотрим его прямо в магазине и положим обратно на полку. Пример неудачный, из прошлого века – сейчас, наверное, уже никто не пользуется бумажными справочниками. Хорошо. Представим себе программу поиска фотографий, которая сначала копирует все фотографии в свою папку и лишь потом начинает искать нужную…

Проблема настолько важна, что редко когда самая сложная задача из Единого государственного экзамена по информатике С4 обходится без нее. Полный балл за эту задачу обычно получают только те программы, которые, читая длинную последовательность, сохраняют в своих данных лишь малую ее часть. На первом занятии мы уже обсуждали одну похожую задачу про крохотный листочек бумаги и карандаш с ластиком.

Дано четное количество чисел. Известно, что их можно разбить на пары так, чтобы суммы во всех парах были одинаковы.

Например, числа 9 2 1 4 6 8 можно разбить на пары (9, 1), (2, 8), (4, 6). Сумма чисел в каждой паре равна 10.

Как быстро, за один проход, определить, можно ли их разбить на пары так, чтобы произведения во всех парах были одинаковы?

Свой ответ можно проверить по Подсказке 5.1.

Правило простое: **если последовательность можно не хранить – ее хранить не нужно**! Из этого правила, кстати, следует эффективность по времени. Раз мы не храним последовательность, то не можем просматривать ее много раз. Все делается за один «проход». А это быстро.

На этом занятии мы рассмотрим такие алгоритмы обработки последовательности, которые позволяют ее не хранить.

Чтение

Прежде всего, о чтении. Читать будем все элементы последовательно в одну переменную. И сразу их обрабатывать.

На этом и следующих занятиях мы будем читать большие объемы данных. Класс Scanner удобный, но работает медленно. Если программа не проходит по времени, возникает Time Limit, то стоит попробовать отказаться от Scanner и использовать StreamTokenizer. Как это сделать рассказывается в конце занятия, есть информация в Справочнике.

Существует два принципиально разных типа условий. Либо сначала задается количество элементов в последовательности, а потом сами элементы. Для такого варианта удобно подходит цикл **for**:

```
int N = in.nextInt(); // чтение количества элементов
for (int i = 0; i < N; i++)
{
     int t = in.nextInt();
     // ...здесь немедленная обработка t
}
```

Иногда последовательность вводится до определенного ограничителя, например, считается, что последовательность заканчивается нулем. В этом случае лучше подходит конструкция **while**:

```
int t = in.nextInt();

while (t != 0) // проверка на ограничитель
{
     // ...здесь немедленная обработка t
     int t = in.nextInt();
}
```

Могут быть и более сложные ограничители. Например, два нуля подряд. Задача с таким ограничением давалась на первом занятии. Решение можно посмотреть в разборе (стр.Ошибка! Закладка не определена.).

Обратите внимание на то, что **во втором случае чтение очередного элемента происходит в конце тела цикла**. Почему? Не теряем ли мы при этом самый первый элемент? (Подсказка 5.2)

Рассмотрим несколько классических алгоритмов.

Сумма элементов

Заведем переменную **s**, проинициализируем ее нулем. И будем добавлять к ней очередной элемент:

```
int N = in.nextInt(); // чтение количества элементов
int s = 0;
for (int i = 0; i < N; i++)
{
     int t = in.nextInt();
     s += t;
}
```

(здесь и далее мы будем иллюстрировать все первым способом чтения, когда сначала задается количество элементов **N**)

Максимум из всех

Точно так же заводим переменную max, и если встречаем элемент больший, чем max, меняем ее значение на новое.

```
int N = in.nextInt(); // чтение количества элементов
int max = ____________;
for (int i = 0; i < N; i++)
{
     int t = in.nextInt();
     if (t > max) max = t;
}
```

В этом коде мы оставили место для самостоятельного заполнения. Как проинициализировать **max**? «Нулем» - неверный ответ! Почему? Для какой последовательности это не сработает?

Подобные алгоритмы называются «жадными». Как только мы находим лучшее, сразу берем его. Это работает далеко не всегда. Допустим, у нас есть плитка разных размеров и нам надо замостить как можно большую площадь данного прямоугольника. Взять самую большую часто неправильно – меньшие могут не влезть, а может оказаться, что маленькими можно замостить больше, чем одной большой!

Максимум из четных

Допустим, по условию в последовательности есть хотя бы одно четное число. И надо найти максимум только из четных. Например, в последовательности 9 2 5 4 3 7 выдать 4.

Эту задачу можно решить двумя способами:

Способ 1. Двумя циклами

Первым циклом идем до первого четного. За это время в переменной max «перебывают» все начальные нечетные. А дальше как в предыдущей задаче

```
int N = in.nextInt(); // чтение количества элементов
int max = in.nextInt();
int i = 0;
while (max % 2 != 0)
{
     max = in.nextInt();
     i++;
}// на выходе из этого цикла в max первое четное число
for (; i < N; i++)// начальное условие цикла можно не указывать
{
```

```
     int t = in.nextInt();
     if (t % 2 == 0 && t > max) max = t; // только среди четных!
}
```

2. Постоянными проверками на начало

Будем в цикле постоянно проверять, нашли ли мы до этого первое четное. Если нет - то перед нами первое четное. Берем его!

```
int N = in.nextInt(); // чтение количества элементов
boolean foundEven = false;
int max = 0;
for (int i = 0; i < N; i++){
     int t = in.nextInt();
     if (t % 2 == 0 && (t > max || !foundEven))//или еще не нашли
     {
          max = t;
          foundEvent = true; // теперь нашли!
     }
}
```

Надо обязательно остановиться и хорошо понять, как работает этот способ.

Очень интересна, и, кстати, тоже часто используется в ЕГЭ, идея поиска второго по величине числа в последовательности.

Второй максимум

Сначала рассмотрим ее в варианте, если оно может совпадать с первым максимумом. Например, для последовательности 2 3 1 6 4 6 3 алгоритм должен выдать 6, потому что число 6 будет на втором месте, если упорядочить последовательность по неубыванию.

Первая идея: «найдем максимум, заменим его на что-нибудь маленькое и найдем максимум еще раз» не хороша. Последовательность по условию этого занятия нужно обрабатывать один раз, а в этом алгоритме нужно обрабатывать ее дважды.

Используем **«метод проталкивания»**.

Заведем две ячейки `max1` и `max2`. Заполним их первыми двумя членами последовательности. При этом упорядочим их так, чтобы `max2` был не больше `max1`.

Далее, если встретим число большее `max1` – заменяем его, «проталкивая» в `max2`. Прежнее значение `max2` при этом теряется.

Если же очередное число оказывается небольше `max1`, но больше `max2`, то просто заменяем `max2` этим числом:

	2, 3	**1**	**6**	**4**	**6**	**3**
max1	3	3	6	6	6	6
max2	2	2	3	4	6	6
			2	3	4	
Комментарий	упорядочим (**max2** ≤ **max1**)	небольше **max2** – не подходит, ничего не делаем	Заменяем **max1**, «проталкивая» его в **max2**	Заменяем **max2**	Заменяем **max2**	небольше **max2** – не подходит, ничего не делаем

Ответ 6.

Вот фрагмент программы, который решает эту задачу

```
int N = in.nextInt(); // чтение количества элементов
int max1 = in.nextInt();
int max2 = in.nextInt();
if (max1 < max2) // упорядочиваем
{
     int t = max1;
     max1 = max2
     max2 = t;
}
for (int i = 0; i < ____; i++){
     int t = in.nextInt();
     if (t > max1) // "проталкиваем"
     {
          max2 = max1;
          max1 = t
     }
     else if (t > max2) // просто меняем max2
     {
          max2 = t;
     }
}
```

Если же нам нужно найти максимальное число, которое строго меньше максимума, то делаем почти так же. Разница будет в инициализации (нужно циклом идти по массиву в поиске начального max2 и немного поменять условие изменения max2. Код фрагмента приведен в Справочнике.

Часто бывает нужно сравнивать соседние элементы последовательности.

В этом случае нужно хранить пару элементов (предыдущий и текущий). Перед чтением очередного элемента текущий становится предыдущим:

```
int prev = in.nextInt(); // prev – предыдущий элемент
for (int i = 0; i < N; i++)
{
     int t = in.nextInt();
     // ...здесь делаем что-нибудь с соседними (prev и t)
     prev = t; // текущий становится предыдущим для следующего
}
```

Немного о задачах

В этом занятии мы не будем разбирать задачи из практической части отдельно. Потому что практически все они решаются небольшими модернизациями алгоритмов, которые мы подробно рассмотрели.

Обязательно прочтите разбор задачи «Старая крепость» после решения, а, возможно, и до него. Эта задача очень важна в идейном плане.

Чтение больших объемов данных

Для чтения большого количества данных на Java лучше использовать буферизованный StreamTokenizer. Удобно сделать его статическим и объявить на уровне главного класса.

```
static StreamTokenizer in = new StreamTokenizer(
         new BufferedReader(new InputStreamReader(System.in)));
```

или

```
static StreamTokenizer in = new StreamTokenizer(
         new BufferedReader(new FileInputStream("input.txt")));
```

для чтения с клавиатуры или из файла.

От Scanner’а этот класс отличается тем, что не имеет специальных функций для чтения значений различных типов. Все данные получаются строковыми. Для сохранения удобства и единообразия можно написать функцию, которая приводит данные к соответствующему типу. Например, для `int`:

```
static int nextInt() throws IOException
{
      in.nextToken();
      return (int)in.nval;
}
```

После этого чтение значений типа int можно делать практически как при использовании Scanner:

```
int a = nextInt();
```

правда, без привычного `in` с точкой в начале.

В качестве примера приведем полностью программу, которая читает последовательность из данного количества элементов и выводит ее сумму.

Пример использования класса StreamTokenizer для быстрого чтения последовательности чисел

```
import java.io.*;
import java.util.*;

public class STExample
{
     static StreamTokenizer in = new StreamTokenizer(
               new BufferedReader(new
InputStreamReader(System.in)));
     static PrintWriter out =
               new PrintWriter(new
                   OutputStreamWriter(System.out));

   public static void main(String[] args) throws IOException
   {
        int N = nextInt();
        int s = 0;
        for (int i = 0; i < N; i++)
        {
            s += nextInt();
        }
        out.println(s);
        out.flush();
   }

   static int nextInt() throws IOException
   {
      in.nextToken();
      return (int)in.nval;
   }
}
```

Задачи

Задача A. Среднее значение последовательности

Определите среднее арифметическое элементов последовательности, завершающейся числом 0.

Число 0 в последовательность не входит. Числа, следующие за нулем, считывать не нужно.

Входные данные

Вводится последовательность целых чисел. Ввод завершается, когда будет введено число 0.

Выходные данные

Выведите одно число - среднее арифметическое элементов последовательности

Пример

Ввод	Вывод
1 7 9 0	5.666666666666667

(количество знаков после точки может быть меньшим)

Задача B. Москва-сортировочная

Источник: Московская командная олимпиада 2004 года

Ежедневно диспетчеру железнодорожной станции «Москва-Сортировочная» приходится переставлять вагоны во многих поездах, чтобы они шли в заданном порядке. Для этого диспетчер может расцепить пришедший на станцию состав в произвольных местах и переставить образовавшиеся сцепки из одного или нескольких вагонов в произвольном порядке. Порядок вагонов в одной сцепке менять нельзя, также нельзя развернуть всю сцепку так, чтобы последний вагон в сцепке оказался первым в ней.

Диспетчер просит вашей помощи в определении того, какое минимальное число соединений между вагонами необходимо расцепить, чтобы переставить вагоны в составе в требуемом порядке.

Формат входных данных

В первой строке входного файла содержится целое число N, ($1 \le N \le 100$). Во второй строке содержится перестановка натуральных чисел от 1 до N (то есть все натуральные числа от 1 до N в некотором порядке). Числа разделяются одним пробелом. Эта перестановка задает номера вагонов в приходящем на станцию составе. Требуется, чтобы в уходящем со станции составе вагоны шли в порядке их номеров.

Формат выходных данных

Программа должна записать в выходной файл единственное целое число, равное минимальному количеству соединений между вагонами, которое нужно расцепить в данном составе, чтобы их можно было переставить по порядку.

Примеры

Входные данные	Выходные данные
4 3 1 2 4	2
5 5 4 3 2 1	4
2 1 2	0

Задача C. Потеряная карточка

Для настольной игры используются карточки с номерами от 1 до *N*. Одна карточка потерялась. Найдите ее, зная номера оставшихся карточек.

Дано число *N*, далее *N-1* номер оставшихся карточек (различные числа от 1 до N). Программа должна вывести номер потерянной карточки.

Ввод	Вывод
5 1 2 3 4	5
4 3 2 4	1

Задача D. Тапочки

Источник: Турнир Архимеда 2007 года

У меня в прихожей стоят в ряд 20 тапочек – 10 левых и 10 правых. Приходя домой, я переобуваюсь и выбираю два тапочка – левый и правый, в которые мне удобнее всего засунуть ноги. Естественно, что левый тапочек должен стоять левее правого, и расстояние (количество других тапочек) между ними должно быть как можно меньше. Напишите программу, которая вычисляет, сколько же тапочек стоит между теми, которые мне удобнее всего надеть.

Входные данные

Вводится последовательность из 10 нулей и 10 единиц, записанных в некотором порядке. Единица соответствует левому тапочку, 0 – правому тапочку. Числа разделены пробелами.

Выходные данные

Программа должна вывести количество тапочек между самыми удобными тапочками, или `-1`, если таких нет.

Пример

Входные данные	Выходные данные	Комментарий
1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0	0	Можно, например, надеть первый (левый) и второй (правый) тапочек, между которыми нет других тапочек.

Задача E. Серебряная медаль

Спортсмен Василий участвовал в соревнованиях по хоккейболу и получил в личном зачете серебряную медаль. Известно, что участники, получившие одинаковое количество очков, награждаются одинаковыми наградами. Известно, что были разыграны золотые, серебряные и бронзовые медали. В задаче не спрашиваются правила хоккейбола. Необходимо только определить, сколько очков набрал Василий.

Формат входного файла

На первой строке дано число N ($2 \le N \le 1000$) количество спортсменов, участвовавших в соревнованиях, на второй N целых чисел – результаты через пробел.

Формат выходного файла

Требуется вывести одно число – результат Василия

Примеры

Ввод	**Вывод**
5 4 3 3 1 2	3
8 1 2 5 3 5 1 1 6	5

Задача F. Ка-штаны

Источник: Московская олимпиада, 7-9 класс 2009 года

Как известно, обычно штаны состоят из двух штанин. Однако собачке нужны, например, уже штаны из 5 штанин (для 4-х лап и хвоста), а сороконожке – штаны с 40 штанинами.
У Пети живет Зверь, у которого M лап. Иногда – когда на улице особенно холодно, чтобы Зверь не простудился, на него бывает нужно надеть несколько штанов, чтобы на каждой лапе было надето по несколько штанин.

Петина мама оставила Пете N штанов, имеющих соответственно K_1, K_2, …, K_N штанин, наказав ему надеть на Зверя их все. Петя хочет надеть на Зверя штаны так, чтобы на самой «утепленной» лапе оказалось как можно меньше штанин, но при этом все оставленные мамой штаны были надеты на зверя. Любые штаны можно надевать на любой набор лап (каждая лапа встречается в наборе не более одного раза).

Помогите ему – напишите программу, которая для каждых штанов укажет, на какие лапы должны быть надеты их штанины. Имейте в виду, что две штанины одних и тех же штанов не могут быть надеты на одну и ту же лапу (в то время как штанины разных штанов могут быть надеты на одну и ту же лапу).

Формат входных данных

Вводится сначала число M, а затем число N ($1 \le M \le 100$, $1 \le N \le 100$). Далее вводятся N чисел K_i, обозначающих число штанин у оставленных мамой штанов ($1 \le K_i \le M$).

Формат выходных данных

Выведите N строк, в i-ой строке должно быть выведено K_i различных чисел, обозначающих номера лап Зверя, на которые должны быть надеты штанины i-ых штанов. Лапы Зверя нумеруются натуральными числами от 1 до M. Если искомых ответов несколько, то выведите любой из них.

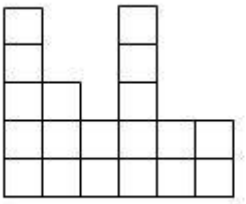
Примеры

Пример ввода	**Пример вывода**	**Комментарий**
4 3 1 2 3	1 1 2 2 3 4	Первые штаны надеты на лапу 1; вторые штаны надеты на лапы 1 и 2; третьи штаны надеты на лапы 2, 3 и 4. Таким образом, на самых «утепленных» лапах (а это лапы 1 и 2) надето по 2 штанины.
4 2 3 2	1 2 3 1 4	Первые штаны надеты на лапы 1, 2 и 3; вторые штаны надеты на лапы 1 и 4. Таким образом, количество штанов на самой «утепленной» лапе (это лапа номер 1) – 2.

Задача G. Старая стена

Вдоль границы двух государств когда-то была построена новая стена. Она была собрана из одинаковых кубических блоков и ее высота по всей длине была одинаковой и равнялась 5 блокам. Много лет этого было достаточно, чтобы удержать соседние королевства от нападения друг на друга. Однако инспекция, посланная одним из королей к стене, обнаружила, что во многих вертикальных рядах один или несколько верхних блоков разрушились или упали.

Инспекция составила отчет, в котором для каждого вертикального ряда блоков указана его нынешняя высота. Военное министерство сразу же заинтересовалось вопросом: где находится самый уязвимый участок стены? Участок стены является уязвимым, если он целиком состоит из подряд идущих рядов, высота которых меньше 5 и ограничен с обеих сторон либо границами стены, либо рядами блоков максимальной высоты.



Например, у стены на рисунке два уязвимых участка (второй и третий ряд; пятый и шестой ряды, считая слева). Длина стены на рисунке равна 6.

Один участок стены считается более уязвимым чем другой, если для его полного восстановления понадобится больше блоков.

Для стены на рисунке второй участок более уязвимый чем первый.

Определите номера рядов, с которого начинается и которым заканчивается самый уязвимый участок стены, а также количество блоков, которые необходимы для его полного восстановления.

Если возможны несколько вариантов ответа, выведите любой.

Исходные данные

сначала вводится число N – длина стены (натуральное, не превышает 1000), затем следует N целых чисел в диапазоне от 0 до 5 – высота соответствующего вертикального ряда. *Гарантируется, что на стене есть хотя бы один уязвимый участок*.

Результат

выведите через пробел номер блока, с которого начинается самый уязвимый участок, затем номер, которым он заканчивается, затем количество блоков, которые нужны для полного восстановления этого участка.

Примеры

Исходные данные	**Результат**
6 5 3 2 5 2 2	5 6 6
10 5 1 5 2 5 3 5 5 0 2	9 10 8

Задача H. Праздничные дни

Источник: VII Нижегородская городская олимпиада 2011 года

В Тридевятом царстве царь был любителем разных заморских традиций. Как прознает, что в другом царстве есть какой-то обычай, сразу думает, как бы его к тридевятым реалиям приспособить.

Вот неделю назад вернулось посольство из Тридесятого царства. И главный посол доложил царю: дескать, придумал Тридесятый царь следующую вещь. Чтобы как-то зарегулировать гуляния народные, повелел он указать определенные дни, и в эти дни устраивать широкие гуляния, а в остальные дни массовые сборища запретить. И с тех пор жизнь в Тридесятом царстве стала прекрасной: гулять так гулять, работать так работать, и все строго по цареву указу.

Понравилась мысль такая царю Тридевятого царства. Подумал он ввести и у себя такие порядки. Собрал царь советников своих, и говорит: подготовьте мне список дней, в которые гулять можно. Только не на год, а на N дней вперед — посмотрим, дескать, что получится; понравится — сделаем круглогодичным.

И вот вчера принесли советники царю список. Но вот незадача: каждый советник свой список приготовил, да еще и обоснование предложил, какой праздник в какой из этих дней надо отмечать. И у всех советников праздники важные, но у всех — разные! Царь думал-думал и решил: а возьмем их все — объединим предложения советников! Если какой-то день есть в списке хотя бы одного советника, то объявим этот день праздничным, и пускай народ гуляет! Глядишь, и не будет недовольных.

Только одна проблема осталась: некоторые дни оказались в списках сразу у нескольких советников. Но царь и тут нашел выход: перенесем некоторые праздники на более поздние дни, так, чтобы в каждый день получался только один праздник, и переносы были бы как можно короче.

Пусть, например, четыре советника сразу предложили сделать некоторый день (пускай день 5) праздничным. Тогда перенесем три из этих четырех праздников на дни 6, 7 и 8 — так, что праздничными будут дни с 5 по 8 включительно. А если оказывается, что, например, день 7 тоже предложен в качестве праздничного кем-нибудь из советников, то перенесем этот праздник еще дальше — на день 9.

Напишите программу, которая, зная предложения советников, определит, какие дни будут праздничными, а какие нет. Не забывайте, что праздники можно переносить только на более поздние дни; на более ранние переносить нельзя.

Формат входных данных

На первой строке входного файла находится одно число N — количество дней, на которые царь хочет произвести планировку праздников.

На второй строке входного файла находятся N неотрицательных целых чисел — для каждого дня указано, сколько советников предложили считать его праздничным.

Гарантируется, что $1 \le N \le 100000$, и что сумма всех чисел во второй строке входного файла не превосходит 100000.

Формат выходных данных

В выходной файл выведите одну строку, состоящую из символов «+» или «–». «+» обозначайте праздничный день, «–» — непраздничный. Выведите как минимум *N* символов — по одному для каждого из дней, на которые проводится планирование. Но если праздники приходится переносить на дни после *N*-го (что допустимо), то выведите больше символов — до последнего праздничного дня.

Символы разделяйте пробелами.

Примеры

Ввод	**Вывод**
5 0 3 0 0 0	– + + + –
10 0 4 0 2 0 0 0 0 1 0	– + + + + + + – + –
3 0 3 0	– + + +

Задача I. Наибольшее произведение

Источник: Московская олимпиада 2004 года

Дано *N* целых чисел. Требуется выбрать из них три таких числа, произведение которых максимально.

Формат входных данных

Во входном файле записано сначала число *N* — количество чисел в последовательности ($3 \le N \le 10^6$). Далее записана сама последовательность: *N* целых чисел, по модулю не превышающих 30000.

Формат выходных данных

В выходной файл выведите три искомых числа в любом порядке. Если существует несколько различных троек чисел, дающих максимальное произведение, то выведите любую из них.

Примеры

Входные данные	**Выходные данные**
9 3 5 1 7 9 0 9 -3 10	9 10 9
3 -5 -30000 -12	-5 -30000 -12

Задача J. Отрезок с максимальной суммой

Дана последовательность целых чисел. Найти отрезок этого массива с максимальной суммой.

Входные данные

В первой строке дано натуральное число n ($1 \le n \le 10^5$) — размер массива. Во второй строке через пробел перечислены элемента массива. Числа не превышают 10^4.

Выходные данные

Выведите три числа — индекс начала отрезка, индекс конца и саму максимальную сумму. Массив индексируется с единицы. Если ответов несколько — выведите любой.

Примеры

Входные данные	Выходные данные
5 -1 2 3 -2 5	2 5 8

Пояснение: в данной последовательности отрезок с максимальной суммой это 2 3 -2 5. Он начинается со второго элемента, заканчивается 5. Сумма его элементов - 8.

Подсказки и решения. 5 Занятие.

5.1

Мысленно расположим числа по неубыванию. Равные суммы образуют пары «с краев» - первое-последнее, второе-предпоследнее и т.д. Но ведь равные произведения при таком расположении должны образовывать те же пары! А значит либо все числа равны между собой, либо половина одних чисел и половина других. Это легко проверить за один проход.

5.2

Нет, не теряем. Ведь первый элемент читается еще до цикла и тут же обрабатывается. А в конце цикла читается новый. Он обрабатывается на следующей итерации. Если первым элементом будет 0, то мы вообще не войдем в цикл, как и должно быть.

Разбор. Занятие 5

5A. Среднее значение последовательности

Для того, чтобы посчитать среднее арифметическое, нужно посчитать количество элементов и их сумму. И выдать ответ – сумму, деленную на количество.

Заведем переменные для суммы и количества.

```
int s = 0, n = 0;
```

И циклом, как разбирается в тексте занятия, посчитаем их:

```
int t = in.nextInt();

for (t != 0) // проверка на ограничитель
{
     s += t;
     n += 1;
     t = in.nextInt();
}
```

при выводе ответа нужно учесть, что нам нужно получить нецелое число, поэтому при делении приводим `s` к типу double.

```
out.println((double)s / n);
```

(Можно было сразу заводить `s` как вещественное `double`.)

5B. Москва-сортировочная

Заметим, что если вагоны не в порядке их по условию обязательно надо расцеплять. «В порядке» означает, что следующий вагон больше предыдущего ровно на единицу.

Посчитаем количество таких мест. Для этого надо «помнить» предыдущий вагон, поэтому используем две переменных, как показано в конце занятия. Единственное, что надо поставить в цикл – сравнение prev и t:

```
int N = in.nextInt(); // чтение количества вагонов
int s = 0;
int r = 0; // количество необходимых расцепок
int prev = in.nextInt(); // prev – "предыдущий вагон"
for (int i = 0; i < N; i++)
{
     int t = in.nextInt();
     if ( t != prev + 1) r++;
     prev = t; // к следующему вагону
}
```

5C. Потерянная карточка

Это красивая задача на идею. Нужно догадаться, что требуемый номер можно получить, если вычесть из суммы всех чисел от 1 до *N* сумму оставшихся (*N*-1 карточек).

Интересно, что можно посчитать обе суммы одним циклом:

```
int N = in.nextInt(); // количество карточек
int s = 0; // сумма оставшихся
int sall = 0; // сумма всех
for (int i = 1; i <= N - 1; i++)// удобно начинать с единицы
{
     int t = in.nextInt();
     s += t;
     sall += i;
}
sall += N; // добавляем в сумму всех карточку с номером N
// ...выводим разность в ответ
```

5D. Тапочки

Можно, конечно, долго пытаться искать минимальное расстояние, но…

Оказывается, что в задаче возможны только ответы -1 и 0, причем -1 только в том случае, если сначала идут все правые!

Достаточно прочитать только первые 10 элементов, найти их сумму. Если она равна нулю, ответ -1, иначе 0!

5E. Серебряная медаль

Единственное, что хочется написать в разборе: «попробуйте написать решение самостоятельно, не подглядывая в Справочник или хотя бы не списывая оттуда код целиком.»

5F. Ка-штаны

Это задача на генерацию последовательности. Будем очередную штанину надевать на очередную лапу, причем расположим лапы «по кругу» - после последней будем надевать снова на первую.

```
int paw = 1; // до цикла первую штанину на первую лапу!
// ...в обычном цикле чтения после чтения очередного t
for (int j = 0; j < t; j++) // надеваем t штанин
{
     out.print (paw + " "); // разделяем пробелами
     paw++;
     if (paw > M) paw = 1;
     // или без if остатком paw = paw % (M + 1) + 1
}
out.println(); // надели очередные штаны – на новую строку!
```

Заметим, что для запутывания участников в тестах приведена другая, хотя и верная, последовательность действий.

5G. Праздники

Задача на моделирование. Нужно просто «пройтись и сделать, что требуется».

Достаточно хранить количество дней, которые еще надо сделать праздничными (`nhol`), и, если на текущий момент их больше нуля, выводить «+», иначе «-».

По условию нужно вывести весь «остаток праздничных дней». Это можно сделать либо дополнительным циклом, либо просто сделать условие цикла таким:

```
while(nhol > 0 || i < N)
```

5H. Старая стена

Прежде всего заметим, что участки с малой защенностью (будем говорить просто «участки») не совпадают. Они отгорожены друг от друга либо высокими стенами либо краями. Поэтому можно **использовать идею так называемого «конечного автомата»**.

Будем хранить характеристики худшего участка. Сначала можно им придать, например, такие значения:

```
int needWorst = 0; // нужно блоков
int startWorst = -1; // начало худшего
int endWorst = -1; // конец худшего
```

Заведем переменную, в которой будем хранить «состояние».

В процессе чтения последовательности оно может быть «в участке» и «не в участке». Сначала мы «не в участке»:

```
int state = 0; // 0 – не в участке, 1 - в участке
```

Будем хранить также характеристики текущего участка:

```
int need = 0; // нужно блоков
int start = -1; // начало худшего
```

В процессе чтения мы можем встретить

a) высокий ряд
b) низкий ряд
c) конец последовательности – дошли до конца

Приведем таблицу переходов и действий при переходах из одного состояния в другое

Что встретили **В каком состоянии**		**высокий ряд**	**низкий ряд**	**конец последовательности**
0	Не в участке	Ничего интересного	Участок начался: запоминаем начало, количество блоков и переходим в состояние 1	Ничего интересного
1	В участке	Участок закончился: Если насчитаны худшие характеристики – запоминаем их (конец худшего участка – это номер текущего ряда плюс 1). Переходим в состояние 0	Участок продолжается: исправляем количество блоков (добавляем к переменой need значение 5-t)	Участок закончился: исправляем количество блоков (добавляем к переменой need значение 5-t) Если насчитаны худшие характеристики – запоминаем их (конец – это номер текущего ряда

				плюс 1). Переходим в состояние 0

Реализуется этот конечный автомат условными операторами:

```
// в цикле чтения последовательности
// ...после чтения очредного t:
if ((t == 5) && state == 1)) // участок закончился высоким рядом
{
     if (needWorst < need)
     {
          needWorst = need;
          startWorst = start;
          endWorst = i; // (заметим, что i равно
                        // номеру текущего ряда минус один)
          state = 0; // переходим в состояние 0
     }
// ...остальные переходы – аналогично
```

5I. Наибольшее произведение

Это технически непростая задача с Московской олимпиады. Нужно найти три максимума и два минимума.

Приведем авторский разбор (с адаптацией к языку Java), хотя во многом он повторяет идеи, изложенные в тексте занятия.

Автор: С.В. Шедов

Сначала попробуем решить задачу очевидным способом, который сразу приходит на ум. Переберем все тройки чисел и выберем из них такую, которая имеет максимальное произведение.

В этом решении существуют две проблемы. Во-первых, оно успевает сработать примерно только при $N < 100$ (Нетрудно подсчитать, что количество раз, которое выполняется тело цикла (количество итераций цикла) равно $N*N*N$ или N^3. Для подсчета времени работы программы удобно считать, что за 1 секунду выполняется порядка 1 миллиона итераций цикла (разумеется, если в теле цикла содержатся вызовы функций или другие циклы, то эта оценка неверна). *(Сейчас, в 2013 году используют другие ограничения. Считается «нормой» выполнение 200 млн. операций в секунду.)*

Однако есть более красивое решение, которое полностью решает задачу с учетом ограничений, поставленных в задаче и при этом не прибегает ни к каким хитростям. Рассмотрим его.

В задаче нам необходимо найти такие три числа, произведение которых максимально. То есть, какие бы другие три числа мы не выбирали, их произведение будет всегда меньше или равно максимальному. Теперь давайте сообразим, какие именно числа в последовательности могут давать максимальное произведение. Первое, что приходит на ум – три максимальных числа последовательности. Для последовательностей с неотрицательными элементами это действительно так.

Рассмотрим пример:

3	1	5	0	***9***	4	***6***	2	***6***	6

Когда все числа в последовательности неотрицательны, то максимальное произведение дадут именно три максимальных элемента. В нашем примере это `9*6*6=324`.

Остается понять, как искать три максимальных элемента. Алгоритм нахождения максимального элемента массива широко известен и не требует пояснений. Но нам надо найти не только максимальный элемент, но и «второй» и «третий» максимумы. В нашем примере первый максимум = 9, второй (следующий по значению) = 6, третий = 6. Обратите внимание, что максимумы могут совпадать по значению.

К счастью, задача поиска трех максимумов решается несложно. Пусть в переменных max1, max2, max3 хранятся первый, второй и третий максимум соответственно. Присвоим им начальные значения, равные –30000. В процессе работы программы они будут либо улучшены, либо оставлены без изменений (в случае, если последовательность состоит только из минимально возможных чисел –30000).

Воспользуемся идеей «проталкивания сверху вниз» очередного элемента в текущие три максимума. Снова обратимся к нашему примеру. Пусть мы уже просмотрели четыре элемента последовательности и правильно заполнили переменные max1, max2 и max3.

max1	max2	max3
5	3	1

Мы считали из входного файла очередное число последовательности ***k*=9**. Сначала сравним его с переменной max1.

```
if (k > max1)
{
     max3 = max2;
     max2 = max1;
     max1 = k;
}
```

Поскольку 9 > 5, то произведем «проталкивание» нового максимума – запишем ***k*** в max1, а в max2 – старое значение max1 (ведь оно было больше или по крайней мере равно max2!). В max3 запишется старое значение max2, прежнее значение max3 при этом теряется – хранить его нет больше смысла.

k	max1	max2	max3
9	5	3	1

Таким образом, мы получим:

max1	max2	max3
9	5	3

Если окажется, что ***k*** ≤ max1, тогда следует его сравнить с max2.

```
if (k > max2)
{
      max3 = max2;
      max2 = k;
}
```

Например, для 7го элемента нашего примера, ***k*** = 6:

k	max1	max2	max3
6	9	5	3

Мы не изменяем max1, вместо max2 записывается ***k***, а на место max3 становится прежний max2.

В противном случае сравнивается ***k*** и max3.

```
if (k > max3) max3 = k;
```

Можно реализовывать не «проталкивание сверху вниз», а «проталкивание снизу вверх». Программная логика в этом случае претерпит лишь небольшие изменения:

```
if (k > max3) max3 = k;
else if (k > max2)
{
     max3 = max2;
     max2 = k;
}
else if (k > max1)
{
     max2 = max1;
     max1 = k;
}
```

Но у нас в последовательности могут быть еще и отрицательные числа! Произведение двух отрицательных чисел положительно и поэтому необходимо найти два минимальных отрицательных числа. Произведем поиск первого и второго минимума min1 и min2 в последовательности аналогичным методом.

Теперь сравним два произведения: min1*min2 и max2*max3. Нам надо выбрать максимальное из них. Очевидно, что максимальным произведением из трех элементов будет максимальный элемент последовательности max1, умноженный на максимум из min1*min2 и max2*max3.

Однако и это еще не все! Тем и замечательны олимпиадные задачи, что мы должны учитывать все возможные случаи, которые допускают ограничения на входные данные. А что будет, если все числа в последовательности отрицательные? Тогда, в случае min1 * min2 > max2 * max3 наш алгоритм найдет далеко не максимальное произведение. Ясно, что в этом случае ответом задачи будут просто три максимальных элемента массива, так как максимальным произведением (отрицательным!) будет max1 * max2 * max3.

Все остальные случаи полностью покрываются нашим алгоритмом. В том числе и различные случаи с нулем. В случае последовательности из отрицательных элементов и нескольких нулей максимальное произведение будет равно 0, но 0 в этом случае в нашем алгоритме обязательно попадет в max1, а остальные два элемента значения играть не будут. Если же в последовательности есть хотя бы один положительный элемент, то max1 никогда не будет равен нулю, поэтому если нулю оказались равны max2 и max3, то при наличии отрицательных элементов максимальное произведение будет формироваться из min1, min2 и max1.

Обратите внимание на то, что мы не создаем массив и не храним последовательность элементов в памяти. Поскольку задача решается в один проход по массиву, то для нашего алгоритма этого и не требуется – в каждый момент времени нам требуется знать только один текущий элемент последовательности. Мы обрабатываем его, а затем «забываем».

5J. Отрезок с максимальной суммой

Мгновенно решившим задачу раскроем тайну мироздания. Бывают еще отрицательные числа! Остальным (и только им!) можно продолжить чтение.

Кажется просто невероятным, что эту задачу можно решить за один проход по последовательности!

Сразу скажем, что есть один особый случай – все отрицательные. Это легко проверить, а в конце выдать ответ – отрезок содержащий единственное максимальное по значению отрицательное.

Будем считать, что в последовательности есть хотя бы один неотрицательный элемент.

Эта задача идейно похожа на задачу «Старая крепость». С одной, но огромной разницей: отрезки здесь могут пересекаться.

Но все-таки пересекаться они могут «не совсем». Подходящие отрезки могут быть только вложенными в друг друга, причем левые концы при этом у них должны совпадать. Поясним на примере.

`-3 3 1 -2 -4 7 -2 3 -1`

Отрезок должен начинаться с неотрицательного числа. Потому что тогда лучше его, отрицательное, выкинуть – результат будет лучше. Плохо начинать с -3. Лучше его пропустить и начать сразу с трех! А положительное число в качестве начала всегда подойдет. Если подойдет (окажется лучшим) отрезок, начиная с тройки, нам нет смысла его сужать, начиная с 1 (будет меньше, а значит хуже!)

Но ведь подобная ситуация возникает, когда отрицательная сумма возникает в левой части отрезка! Скажем, мы хотим начать с 3.

3 + 1 = 4 лучше,

4 - 2 = 2 терпимо, мы пока в плюсе

2 – 4 = -2 – ушли в минус.

Значит, начинать с 3 смысла нет, лучше все пропустить, и начать уже с 7.

Таким образом, алгоритм получается очень простым. Накапливаем текущую сумму, и если она становится больше уже найденной максимальной – корректируем ответ, а если она становится меньше нуля – идем дальше до следующего неотрицательного.

Для иллюстрации построим подробную табличку действий алгоритма по нашему примеру:

«Претенденты»

Лучшие «претенденты»

Номера элементов									Что происходит	Summ	Start	End	Best
1	2	3	4	5	6	7	8	9		0	0	0	0 0 0
-3	3	1	-2	-4	7	-2	3	-1	Отрицательное пропускаем	0	0	0	0 0 0
-3	3	1	-2	-4	7	-2	3	-1	Начался «претендент»	3	2	2	3 2 2
-3	3	1	-2	-4	7	-2	3	-1	«Претендент» «растет»	4	2	3	4 2 3
-3	3	1	-2	-4	7	-2	3	-1	«Претендент» «растет»	2	2	4	4 2 3
-3	3	1	-2	-4	7	-2	3	-1	«Претендент» закончился	2	2	4	4 2 3
-3	3	1	-2	-4	7	-2	3	-1	Начался «претендент»	7	5	5	7 5 5
-3	3	1	-2	-4	7	-2	3	-1	«Претендент» «растет»	5	5	7	7 5 5
-3	3	1	-2	-4	7	-2	3	-1	«Претендент» «растет»	8	5	8	8 5 8
-3	3	1	-2	-4	7	-2	3	-1	«Претендент» закончился	8	5	8	8 5 8

Еще раз повторим, что эта часть работает корректно, если в последовательности есть хотя бы одно неотрицательное число. Занятие 5.Справочник

Чтение больших объемов данных

`PrintWriter` подходит для всех случаев и работает достаточно быстро. Но его метод `printf` работает медленно; также медленно работают вызовы типа `println(a + " " + b)`. Выводите по одной переменной за раз, и тогда вы добьетесь максимальной эффективности.

Лучше для этого использовать буферизованный StreamTokenizer. Удобно сделать его статическим и объявить на уровне главного класса.

```
static StreamTokenizer in = new StreamTokenizer(
         new BufferedReader(new InputStreamReader(System.in)));
```

или

```
static StreamTokenizer in = new StreamTokenizer(
         new BufferedReader(new FileInputStream("input.txt")));
```

для чтения с клавиатуры или из файла.

От Scanner'а этот класс отличается тем, что не имеет специальных функций для чтения значений различных типов. Все данные получаются строковыми. Для сохранения удобства и единообразия можно написать функцию, которая приводит данные к соответствующему типу. Например, для `int`:

```
static int nextInt() throws IOException
{
      in.nextToken();
      return (int)in.nval;
}
```

После этого чтение значений типа int можно делать практически как при использовании `Scanner`:

```
int a = nextInt();
```

правда, без привычного in с точкой в начале.

В качестве примера приведем полностью программу, которая читает последовательность из данного количества элементов и выводит ее сумму.

Чтение последовательности, дано количество элементов

```
int N = in.nextInt(); // чтение количества элементов
for (int i = 0; i < N; i++)
{
     int t = in.nextInt();
     // ...здесь немедленная обработка t
}
```

Чтение последовательности до нуля

```
int t = in.nextInt();

for (t != 0) // проверка на ограничитель
{
     // ...здесь немедленная обработка t
     int t = in.nextInt();
}
```

Могут быть и более сложные ограничители. Например, два нуля подряд. Эта задача давалась в первом занятии. Решение можно посмотреть в разборе.

Сумма элементов

Заведем переменную `s`, проинициализируем ее нулем. И будем добавлять к ней очередной элемент:

```
int N = in.nextInt(); // чтение количества элементов
int s = 0;
for (int i = 0; i < N; i++)
{
     int t = in.nextInt();
     s += t;
}
```

(здесь и далее мы будем иллюстрировать все первым способом чтения)

Максимум из всех

```
int N = in.nextInt(); // чтение количества элементов
int max = in.nextInt();
for (int i = 0; i < N; i++)
{
     int t = in.nextInt();
     if (t > max) max = t;
}
```

Максимум из четных двумя циклами

```
int N = in.nextInt(); // чтение количества элементов
int max = in.nextInt();
int i = 0;
while (max % 2 != 0)
{
     max = in.nextInt();
     i++;
}// на выходе из этого цикла в max первое четное число
for (; i < N; i++)// начальные условия цикла можно не указывать
{
     int t = in.nextInt();
     if (t % 2 == 0 && t > max) max = t; // только среди четных!
}
```

Второй максимум с повторениями

```
int N = in.nextInt(); // чтение количества элементов
int max1 = in.nextInt();
int max2 = in.nextInt();
if (max1 < max2) // упорядочиваем
{
     int t = max1;
     max1 = max2
     max2 = t;
}
for (int i = 0; i < N - 2; i++){
     int t = in.nextInt();
     if (t > max1) // "проталкиваем"
     {
          max2 = max1;
          max1 = t
     }
     else if (t > max2) // просто меняем max2
     {
          max2 = t;
     }
```

```
}
```

Второй максимум без повторений

```
int N = in.nextInt(); // чтение количества элементов
int max1 = in.nextInt();
int max2 = in.nextInt();
int k = 2;// счетчик прочитанных элементов до основного цикла
//в массиве должны быть хотя бы два различных элемента
while(max2 == max1)
{
     max2 = in.nextInt();
     k++;
}
if (max1 < max2) // упорядочиваем
{
     int t = max1;
     max1 = max2
     max2 = t;
}
for (int i = 0; i < N - k; i++){
     int t = in.nextInt();
     if (t > max1) // "проталкиваем"
     {
          max2 = max1;
          max1 = t
     }
     else if (t > max2 && t != max1) // просто меняем max2
     {
          max2 = t;
     }
}
```