Занятие №6. Массивы

При решении многих задач возникает необходимость хранить всю последовательность или ее часть. Конечно, можно завести много переменных для этого: a, b, c и т.д. Но в этом случае **а)** быстро кончатся буквы, **б)** неизвестно, сколько именно этих переменных нам понадобится, причем от запуска к запуску программы, это количество должно меняться, но главное **в)** для обработки последовательности нельзя будет использовать циклы!

Для хранения большого количества данных *одного типа* используют массивы. Массив – это переменная, в которой хранится много значений. При создании массива указывается его размер.

Создание массива

Массивы в языке Java – это объекты. Поэтому чаще всего они создаются операцией new:

```
int[] arr = new int[10];
```

В языках C, C++ квадратные скобки в левой части ставятся после имени переменной. В Java можно использовать оба варианта синтаксиса:

```
int arr[] = new int[10];
```

Создается массив с именем arr размером на 10 ячеек типа int.

индексы (номера)	0	1	2	3	4	5	6	7	8	9
значения	0	0	0	0	0	0	0	0	0	0

Обратите внимание, ячейки-элементы массива нумеруются с нуля. В Java ячейки массива по умолчанию инициализируются нулями, а массивах объектов значениями null. (В некоторых языках программирования это не так.) Иногда нужно инициализировать массив конкретными ненулевыми значениями. Для этого удобно использовать объявление с инициализацией:

Пусть n = 5, тогда код

```
double[] arr = new double[n]{2.0, 3.5, -1};
```

создает массив с именем arr размером на 5 ячеек. При этом оставшиеся ячейки, которым «не хватило инициализаторов» заполняются нулями.

индексы (номера)	0	1	2	3	4
значения	2.0	3.5	-1.0	0.0	0.0

Еще раз заметим, что можно создать массивы «чего угодно», объектов любых типов. Например, можно создать массив строк:

String[] str = new String{"ab", "def", "a bc d"};

При создании массива с инициализацией можно не указывать размер. В данном случае будет создан массив str из трех строк.

Возможно даже создать массив массивов, мы сделаем это на занятии, посвященном двумерным массивам. Для простоты изложения во всех следующих примерах мы будем использовать массив с именем «**a**» типа **int**.

С массивом после создания работают в основном «поэлементно». Например, для того, чтобы «считать массив» нужно считать каждый его элемент. Многие типовые операции устроены практически так же, как мы обсуждали на занятии «Однопроходные алгоритмы». С той разницей, что мы работаем в цикле не с одной переменной (t), а последовательно с каждой ячейкой массива.

Приведем несколько примеров.

Ввод (считывание) массива из N элементов

```
for (int i = 0; i < N; i++)
{
     ___ = in.nextInt();
}
```

Вывод всех элементов массива

```
for (int i = 0; i < N; i++)
{
     out.print( __________);
}
```

Поиск максимума

```
int max = ____;
for (int i = 1; i < N; i++)
{
     if (________)
     {
          ________;
```

```
     }
}
```

Важной, даже более важной, чем поиск максимального значения в массиве, является задача *поиска места его в массиве*. Ведь, зная индекс, мы сразу можем найти значение, а по значению найти индекс сложно – нужно заново проходить по массиву и искать!

Поиск индекса максимального

Напишите его практически самостоятельно:

```
int imax = ____;
_____________;
for (int i = 1; i < N; i++)
{
     if (__________)
     {
          _________;
          _________;
}
}
```

Отметим, что в этой программе можно не хранить текущий максимум в отдельной переменной. И вместо него постоянно использовать a[imax]. Таким образом, в программе выше две строки с подчеркиваниями оказываются вообще лишними! Сверьте свой код со Справочником.

Часто бывает нужно определить, есть ли в массиве заданное число. Это удобнее всего писать циклом while.

Поиск индекса заданного числа в массиве

```
int r = -1; // индекс элемента, равного x
int i == 0;
while (_____ && i < N)
{
     if (a[i] == x) r = i;
     i++;
}
```

Если r после цикла останется -1 – элемент не найден.

Чтобы не рассматривать отдельно конец последовательности, когда элемент в массиве отсутствует, можно дополнить его фиктивным **«барьерным» элементом** (для этого в массиве должна быть обязательно хотя бы одна «лишняя» ячейка):

```
a[N] = x;
int r == 0;
while (a[r] != x)
{
     r++;
}
```

Теперь мы в любом случае «найдем» **x**. Отсутствие элемента теперь не особый случай – **x** не обнаружен в последовательности, если r станет равным n.

Рассмотрим теперь несколько задач, в которых использование массива необходимо, то есть последовательность нужно хранить.

Вывод массива в обратном порядке

```
for (int i = _____; i _____; i++)
{
     out.print(a[i] + " ");
}
```

(Сверьте заполнение с Подсказкой 6.1)

Сложнее «перевернуть» массив. Именно переставить его элементы, так чтобы нулевой элемент встал на *N*-1-е место, а *N*-1-й на нулевое. (И в жизни обычно бывает проще сказать, чем сделать!).

a[0]↔a[N-1]

a[1]↔a[N-2]

…

a[i]↔a[____]

(сверьтесь с Подсказкой 6.2)

Обменять местами две переменных, скажем a и b можно при помощи третьей. Обычно используют аналогию со стаканами. В одном стакане чай, в другом молока. Поменять их содержимое при помощи третьего стакана:

```
int t = a;
a = b; // содержимое a надежно сохранено в t
b = t;
```

Напишем программу:

```
for (int i = 0; i < N; i++)
```

```
{
     int t = a[i];
     a[i] = a[N - i - 1];
     a[N – i - 1] = t;
}
```

Но оказывается, что она не работает!

Как именно не работает и как это исправить? (Подсказка 6.3)

Для того, чтобы избежать трудностей – придумывания формулы, нестандартного условия цикла, можно воспользоваться интересным приемом – **ввести два указателя**.

Заведем две переменных left и right, в которых будем хранить индексы обмениваемых ячеек.

Они вначале будут равны 0 и N-1 соответственно.

```
int left = 0;
int right = N-1;
```

В цикле будем обменивать ячейки с индексами left и right и переходить к следующим:

```
_____________
_____________
_____________
_____________
_____________
```

(Подсказка 6.4)

Удобно использовать цикл while:

```
while(_________________)
{
     int t = a[left];
     a[left] = a[right];
     a[right] = t;
}
```

В этом подходе условие цикла естественно и понятно! Посмотрите полный код в Справочнике.

Еще одной удивительной идеей, связанной с массивами, о которой надо знать обязательно, является идея косвенной адресации.

Косвенная адресация

Значения элементов одного массива используются как индексы ячеек в другом.

Рассмотрим эту идею на решении задачи «Цифры». Приведем ее условие прямо здесь.

На вход программе подается последовательность чисел от 1 до 9, заканчивающаяся нулем. Всего будет введено не более 100000 чисел.

Подсчитайте в этой последовательности количество единиц, количество двоек, количество троек и т.д. и выдайте результат. В выходных данных всегда должно быть 9 чисел.

Можно завести 10 переменных и написать в цикле прохода по массиву что-нибудь в стиле

```
if (a[i] == 0) b++;
if (a[i] == 1) c++;
if (a[i] == 2) d++;
// ................
if (a[i] == 8) j++;
if (a[i] == 9) k++;
```

И затем выводим все переменные от b до k.

Это работает, но решение совсем некрасиво. Нам надо помнить, что в переменой g храниться количество… попробуйте быстро сказать каких цифр! Можно, конечно, называть переменные не b, c, d и так далее, а c0, c1, c2,… c9. Это сделает код более читаемым, но не исправит ситуацию с множеством практически одинаковых строк. Мы по-прежнему не можем использовать цикл! Что делать, если массив заполнен не цифрами, а числами от 0 до 100, и надо дать ответ про количества каждого из этих чисел?

Идея в том, чтобы завести второй массив count на 10 ячеек. И в каждой ячейке хранить количество соответствующей ее индексу цифры. Код станет таким:

```
if (a[i] == 0) count[0]++;
if (a[i] == 1) count[1]++;
if (a[i] == 2) count[2]++;
// ......................
if (a[i] == 8) count[8]++;
if (a[i] == 9) count[9]++;
```

А теперь заметим, что все эти строчки можно заменить одной.

В которой не будет даже условия:

```
count[a[i]]++;
```

Этот код достаточно сложно выглядит. Обязательно остановитесь и постарайтесь хорошо понять, как он работает.

Значение ячейки массива «a» берется в качестве индекса ячейки массива count, которая увеличивается.

Идея косвенной адресации часто используется и на уровне процессора.

Например, команда **`mov ax,[bx]`** на языке ассемблера – языке низкого уровня, практически уровня машинных команд, означает *загрузить в регистр (переменную)* **`ax`** *значение из ячейки с адресом (индексом), хранящемся в* **`bx`**.

И, конечно, без хранения последовательности невозможно выполнить ее сортировку, то есть выдать (переставить) ее элементы в определенном порядке, например, по возрастанию. Об этом мы поговорим на следующем занятии.

Остается повторить материал Занятия №2.

Массивы в Java – это объекты. Работа с ними происходит при помощи ссылок. Поэтому множно, например, можно передавать массив в функцию и там его изменять. Именно переданный массив, а не его копию.

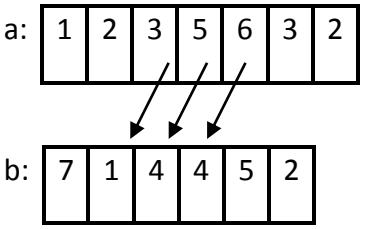
Копировать массивы можно поэлементно, используя цикл. Но удобнее использовать для это статическую функцию arraycopy класса System.

```
System.arraycopy(src, from, dest, to, length)
```

Скопирует из массива **`src`** в массив **`dest`** **`length`** символов, начиная с **`from`**. Причем скопирует их начиная с **`to`**.

У этой функции много параметров. Для ясности напишем код более подробно:

```
int[] a = new int[]{1, 2, 3, 5, 6, 3, 2};
int[] b = new int[]{7, 1, 4, 4, 5, 2}
System.arraycopy(a, 2, b, 3, 1);
```



Из массива a, начиная со второго (третьего по счету) в массив b скопируются 3 символа. Причем, они будут размещены с первого (второго по счету) места в массиве b.

Таким образом, массив b станет таким: 7, 3, 5, 6, 5, 2

Задачи

Задача A. Шеренга

Петя Васичкин перешёл в другую школу. На уроке физкультуры ему понадобилось определить своё место в строю...

Формат входного файла

Сначала вводится *N* – количество человек в классе. Затем невозрастающая последовательность из *N* чисел, означающих рост каждого человека в строю. После этого *X* – рост Пети. Все числа во входных данных натуральные и не превышают 200.

Формат выходного файла

Требуется вывести номер, под которым Петя должен встать в строй. Если в строю есть люди с одинаковым ростом, таким же, как у Пети, то он должен встать после них.

Примеры

Ввод	**Вывод**
8 165 163 160 160 157 157 155 154 162	3
8 165 163 160 160 157 157 155 154 160	5

Задача B. Цифры

На вход программе подается последовательность чисел от 1 до 9, заканчивающаяся нулем. Всего будет введено не более 100000 чисел.

Подсчитайте в этой последовательности количество единиц, количество двоек, количество троек и т.д. и выдайте результат. В выходных данных всегда должно быть 9 чисел.

Пример

Входные данные	**Выходные данные**
1 1 4 1 5 8 6 3 5 1 0	4 0 1 1 2 1 0 1 0

Задача C. Прыжки с трамплина

Источник: Турнир Архимеда 2010 года

На соревнованиях по прыжкам на лыжах с трамплина техника прыжка оценивается пятью судьями. Каждый судья ставит оценку от 1 до 20, после чего одна наименьшая и одна

наибольшая оценки отбрасываются. Вам нужно написать программу, которая будет демонстрировать результаты прыжка для телетрансляции.

Она должна выводить пять оценок, которые поставили судьи, не меняя их порядка, а затем их сумму, и при этом брать в скобки те оценки, которые не учитываются при расчете суммы

Формат входных данных

На вход подается 5 натуральных чисел от 1 до 20, разделенных пробелом.

Формат выходных данных

Выведите те же числа в том же порядке, взяв в скобки минимальное (а если их несколько – самое левое из них) и максимальное (а если их несколько – самое правое из них) число, а также сумму всех чисел, не взятых в скобки. Все числа (включая сумму) должны быть напечатаны в одной строке и разделены одним пробелом (внутри скобок пробелов быть не должно). Перед суммой должен стоять знак равенства, отделенный слева и справа одним пробелом. Порядок оценок должен быть такой же, как и во входных данных.

Примеры

Входные данные	Выходные данные
1 2 3 4 5	(1) 2 3 4 (5) = 9
10 11 10 11 10	(10) 11 10 (11) 10 = 31

Задача D. Дипломы в папках (6-9)

Источник: Московская олимпиада, 6-9 класс 2011 года

В этом году Иван заканчивает школу и поступает в вуз. За время своей учебы он часто участвовал в олимпиадах по информатике и у него накопилось много дипломов. Иван раскладывал дипломы по папкам совершенно бессистемно, то есть любой диплом мог оказаться в любой из папок. К счастью, Иван помнит, сколько дипломов лежит в каждой из папок.

Иван хочет принести в приемную комиссию выбранного вуза папку, в которой находится диплом Московской олимпиады по программированию (такой диплом у Ивана ровно один). Для того чтобы понять, что в данной папке нужного диплома нет, Ивану нужно просмотреть все дипломы из этой папки. Просмотр одного диплома занимает у него ровно одну секунду и он может мгновенно переходить к просмотру следующей папки после окончания просмотра предыдущей. Порядок просмотра папок Иван может выбирать.

По заданному количеству дипломов в каждой из папок требуется определить, за какое наименьшее время в худшем случае Иван поймет, в какой папке содержится нужный ему диплом.

Входные данные

В первой строке входного файла записано целое число N ($1 \le N \le 100$) - количество папок. Во второй строке записаны N целых чисел $a_1, a_2, ..., a_N$ ($1 \le a_i \le 100$) - количество дипломов в каждой из папок.

Выходные данные

Выведите одно число - минимальное количество секунд, необходимое Ивану в худшем случае для определения того, в какой папке содержится диплом.

Примеры тестов

Входные данные	Выходные данные
2 2 1	1

В примере Иван может просмотреть папку 2 за 1 секунду и, не найдя там диплома, понять, что диплом находится в папке 1.

Если же он найдет диплом в папке 2, то на поиск уйдет также 1 секунда.

Задача E. Уникальные элементы

На вход программе сначала подается значение $n \le 100$ — количество элементов в массиве. В следующей строке входных данных расположены сами элементы массива — целые числа, по модулю не превосходящие 30000. Распечатайте только те значения элементов массива, которые встречаются в нем ровно один раз. Элементы следует распечатывать в том порядке, в котором они встречаются в массиве.

Входные данные	Выходные данные
8 4 3 5 2 5 1 3 5	4 2 1

Задача F. Двойной переворот

Дана последовательность натуральных чисел 1, 2, 3, ..., N ($1 \le N \le 1000$). Необходимо сначала расположить в обратном порядке часть этой последовательности от элемента с номером A до элемента с номером B, а затем от C до D ($A < B$; $C < D$; $1 \le A, B, C, D \le N$).

Формат входного файла

Даны числа *N*, *A*, *B*, *C*, *D*.

Формат выходного файла

Требуется вывести полученную последовательность.

Примеры

Ввод	Вывод
9 2 5 6 9	1 5 4 3 2 9 8 7 6
9 3 6 5 8	1 2 6 5 8 7 3 4 9

Задача G. Списывание

Источник: Московская олимпиада, окружной этап 2010 года

На контрольной работе N учеников сидят в ряд. Для каждого ученика известно, какую оценку он получил бы, если бы писал эту контрольную самостоятельно (оценка — это число от 2 до 5). Однако ученики могут писать контрольную не только самостоятельно, но и списывать у своего соседа, но только если сосед пишет контрольную самостоятельно. В этом случае списывающий получит такую же оценку, какую получит тот, у кого он списал.

А именно (правила применяются строго в указанном порядке):

- Школьники, которые знают материал на 5, будут писать контрольную самостоятельно.
- Школьник, который знает материал на 4, если он сидит рядом с тем, кто знает на 5, будет списывать у него, а в противном случае будет писать самостоятельно.
- Школьник, который знает на 3, если он сидит рядом с тем, кто знает на 5, будет списывать у него. Если среди его соседей знающего на 5 нет, но есть тот, кто знает на 4, и при этом пишет самостоятельно, то троечник будет списывать у него. В противном случае будет писать самостоятельно.
- Аналогично школьник, знающий на 2 — из соседей, которые пишут самостоятельно, выберет того, кто знает лучше, и спишет у него. А если таких нет (или оба его соседа также знают на 2), то будет писать самостоятельно.

Определите, кто какую оценку в итоге получит.

Входные данные

Вводится число N ($1 \le N \le 10$) - количество учеников, и далее последовательность из N чисел, описывающая, кто на какую оценку может написать контрольную, если будет писать самостоятельно.

Выходные данные

Выведите *N* чисел - оценки, которые получат ученики за контрольную.

Пример ввода	Пример вывода	Пояснение
5 5 2 3 4 5	5 5 3 5 5	Первый и пятый ученики будут писать самостоятельно. Второй спишет у первого, а четвертый — у пятого (в итоге также получат пятерки). Третьему не у кого списывать, так как его соседи будут писать работу не самостоятельно.
6 2 2 3 2 2 4	2 3 3 3 4 4	Второй и четвертый спишут у третьего, пятый — у шестого.

Задача H. Последовательности

Источник: Московская олимпиада, окружной этап 2010 года

Рассмотрим последовательности чисел.

Первая последовательность состоит из одного числа *K*.

Каждая следующая последовательность чисел описывает предыдущую по такому правилу.

Просматриваем описываемую последовательность слева направо и разбиваем на отрезки, состоящие из подряд идущих равных чисел (причем все идущие подряд одинаковые числа всегда объединяем в один отрезок). Далее каждый такой отрезок описываем двумя числами — первое число говорит, сколько раз повторяется одно и то же число, второе число говорит, какое именно число повторяется. Записываем эти пары последовательно в соответствии с отрезками слева направо, и получаем новую последовательность (см. примеры ниже).

Например, для *K*=2 последовательности получатся такими:

№	Последовательность	Как ее читать (слова в описании соответствуют числам текущей последовательности слева направо, и описывают предыдущую последовательность)
1	2	Исходная последовательность
2	1 2	Одна «двойка»

3	1 1 1 2	Одна «единица», одна «двойка»
4	3 1 1 2	Три «единицы», одна «двойка»
5	1 3 2 1 1 2	Одна «тройка», две «единицы», одна «двойка»
6	1 1 1 3 1 2 2 1 1 2	Одна «единица», одна «тройка», одна «двойка», две «единицы», одна «двойка»

Напишите программу, которая по исходному числу K напечатает N-ую получающуюся последовательность.

Входные данные

Вводится число *K* (1 ≤ *K* ≤ 9) и число N (1 ≤ *N* ≤ 15).

Выходные данные

Ваша программа должна печатать *N*-ую последовательность, полученную из начальной последовательности, состоящей из одного числа *K*. Числа при выводе следует разделять пробелами.

Пример ввода	**Пример вывода**
2 6	1 1 1 3 1 2 2 1 1 2
2 1	2
1 3	2 1

Задача I. Симметричная последовательность

Источник: Московская олимпиада, 7-9 класс 2006 года

Последовательность чисел назовем симметричной, если она одинаково читается как слева направо, так и справа налево. Например, следующие последовательности являются симметричными:

1 2 3 4 5 4 3 2 1

1 2 1 2 2 1 2 1

Вашей программе будет дана последовательность чисел. Требуется определить, какое минимальное количество и каких чисел надо приписать в конец этой последовательности, чтобы она стала симметричной.

Формат входных данных

Сначала вводится число *N* — количество элементов исходной последовательности ($1 \le N \le 100$). Далее идут *N* чисел — элементы этой последовательности, натуральные числа от 1 до 9.

Формат выходных данных

Выведите сначала число *M* — минимальное количество элементов, которое надо дописать к последовательности, а потом *M* чисел (каждое — от 1 до 9) — числа, которые надо дописать к последовательности.

Примеры

Входные данные	**Выходные данные**
9 1 2 3 4 5 4 3 2 1	0
5 1 2 1 2 2	3 1 2 1
5 1 2 3 4 5	4 4 3 2 1

Задача J. Шарики

В одной компьютерной игре игрок выставляет в линию шарики разных цветов. Когда образуется непрерывная цепочка из трех и более шариков одного цвета, она удаляется из линии. Все шарики при этом сдвигаются друг к другу, и ситуация может повториться.

Напишите программу, которая по данной ситуации определяет, сколько шариков будет сейчас уничтожено. Естественно, непрерывных цепочек из трех и более одноцветных шаров в начальный может быть не более одной.

Формат входного файла

Даны количество шариков в цепочке (не более 1000) и цвета шариков (от 0 до 9, каждому цвету соответствует свое целое число).

Формат выходного файла

Требуется вывести количество шариков, которое будет уничтожено.

Примеры

Ввод	**Вывод**
5 1 3 3 3 2	3
10 3 3 2 1 1 1 2 2 3 3	10

Подсказки и решения. 6 Занятие.

6.1

```
for (int i = N-1 ; i >= 0; i++)
{
     out.print(a[i]+ " ");
}
```

6.2

a[i]↔a[N – i - 1]

6.3

Код выполняет много работы, но в конце все элементы оказываются … на своих изначальных местах! Действительно, массив переворачивается дважды. Например, сначала нулевой элемент меняется с N-1-м, а на последнем шаге N-1-й меняется с нулевым обратно. Чтобы этого не происходило, надо идти циклом до середины, а не до конца.

```
for (int i = 0; i < N/2; i++)
```

6.4

```
int t = a[left];
a[left] = a[right];
a[right] = t;
left++;
right--;
```

6.5

```
k = 0; // в начале поиска для всех элементов
```

Разбор. Занятие 6

6A. Шеренга

Эта задача практически с предыдущего занятия «Однопроходные алгоритмы», если бы не формат входных данных. Здесь сначала вводится последовательность и только после нее число для поиска. Поэтому последовательность необходимо сохранить в массиве.

Заведем массив на элемент больше, чтобы использовать идею «барьерного элемента». В последней ячейке будет ноль и Петя в любом случае найдет свое место в строю, ведь его рост больше нуля.

```
int N = in.nextInt();
int[] h = new int[N+1];
```

и заполняем его стандартным способом:

```
for (int i= 0; i < N; i++)
{
     a[i] = in.nextInt();
}
```

Заводим и читаем X

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

Теперь проходим по массиву пока X <= a[i] (обратите внимание на это условие. Почему именно “меньше или равно”, а не просто «меньше» (Подсказка 6.10)?

```
i = 0;
while(X <= a[i])
{
     i++;
}
```

В ответ выводится i + 1, потому что в условии задачи места нумеруются с единицы.

```
out.println(i + 1);
```

6B. Цифры

Задача подробно разбирается в тексте занятия.

Сам массив (все числа) хранить не нужно. Достаточно читать по одному числу и корректировать массив count.

6C. Прыжки с трамплина

Задачу нужно решать в два этапа. На первом найти индексы «правильных» максимума и минимума, а потом вывести массив, заключив эти элементы в скобки.

Решение задачи поиска максимума и минимума подробно обсуждалась. Посмотрите код в Справочнике. Особенность этой задачи в том, чтобы найти именно *первый* минимум и *последний* максимум. Для этого просто поставим в условие разные знаки (< и >= соответственно). Можно сделать два прохода по массиву, можно искать оба индекса за один.

Дальше нужно вывести массив почти стандартно. Когда доходим до соответствующих индексов, заключаем элементы в скобки:

```
for (int i = 0; i < N; i++)
{
     if (i == imin || i == imax) out.print("("+a[i]+") ");
     else out.print(a[i] + " ");
}
```

1D. Дипломы в папках

Достаточно простая задача, которая предлагалась на «младшей» олимпиаде.

Понятно, что самую толстую папку надо оставить последней. То есть нужно найти сумму всех элементов и вычесть из нее максимум. Заметим, что это можно сделать даже за один проход и без хранения последовательности, то есть без массива!

1E. Уникальные элементы

Задачу можно решать, например, так. Пройдем по массиву и *для каждого* элемента пройдем по массиву, считая количество равных ему. Если оно равно одному – выводить элемент.

```
int k = 0;
for (int i = 0; i < n; i++)// для каждого элемента
{
     ___________
     for (int j = 0; j < n; j++) // проходим по всему массиву
     {
          if (a[i] == a[j]) k++;
     }
     if (k == 1) out.println(a[i]+" "); // если элемент уникален
–                                       // выводим его
}
```

Какая важная строка пропущена? Подсказка 6.5.

6F. Двойной переворот

В этой задаче числа не даны. Нужно их «сгенерировать».

Заполним массив простым циклом:

```
for(int i = 0; i < N; i++)
{
     a[i] = i + 1;
}
```

И перевернуть дважды: сначала от A до B, а потом от C до D.

Переворачивать проще всего, применив метод указателей, который описывается в тексте занятия.

6G. Списывание

По условию этой задаче списывать «по цепочке», то есть троечник не будет списывать у четверочника, который списывает у пятерочника. Поэтому нельзя делать изменения сразу в исходном массиве.

Пусть массив с первоначальными оценками называется «a».

Заведем второй массив (назовем его `b`) на `N` ячеек и будем записывать часть окончательных оценок туда.

Теперь можно просто смоделировать процесс, как написано в условии.

Пройдем по массиву и поставим в массив `b` пятерки тем, которые сидят рядом с отличниками. А в массиве «a» их оценки обнулим (они списывали и другим «не интересны»).

Снова пройдем по массиву и выставим всем, сидящим рядом с четверочниками четверки, а их оценки обнулим. Ту же операцию произведем с троечниками.

Таким образом, в массиве `a` останутся оценки тех, кто не списывал и отличников, а в массиве `b` тех, кто списывал.

Интересной идеей является вывод в ответ просто суммы `a[i]` и `b[i]` для каждого ученика.

Учтите, что нужно либо корректно обрабатывать границы дополнительными условиями, либо использовать барьерные элементы, чтобы у первого и последнего ученика появились соседи. Например, чтобы это были «ученики» с оценкой 0. (Для того, чтобы «добавить» элемент в начало, достаточно считывать оценки в массив, начиная с единицы). Конечно, при этом надо завести оба массива размером `N + 2`.

6H. Последовательность

Хотя нужно сделать ровно то, что просят в условии, это технически сложная задача. Будем несколько раз из одного массива «раскрывать» последовательность в другой, и перекидывать результат обратно в первый массив.

6I. Симметричная последовательность

Предположим, что мы нашли k чисел, которые решают задачу:

k чисел в начале	«середина»	k приписанных чисел

Начало и конец симметричны. При этом «Середина» должна быть сама симметрична.

Видно, чтобы выполнить условие минимальности числа k, «середина» должна быть как можно длиннее. Но ведь она является «концом» данной нам последовательности.

Найдем самый длинный симметричный «конец» и припишем к нему «начало в обратном порядке». Например, для 4 1 2 3 2 – это 2 3 2. Приписываем 1 4.

Решение удобно оформить при помощи функции, проверяющей часть массива на симметричность. Проще всего ее написать при помощи двух указателей:

```
static boolean isSymmetric(int[] a, int left, int right)
{
     while (left < right)
     {
          if (a[left] != a[right]) return false; // сразу выходим
     }
     return true; // вышли из цикла – "ошибок" не найдено
}
```

Основной цикл решения получается очень лаконичным:

```
int k = 0; // чтобы использовать k после цикла,объявляем его вне
for (; k < N; k++)
{
     if (isSymmetric(a, k, N-1)) break;
}
```

Остается приписать **k** символов в обратном порядке (минимально 0, максимально **N –1**)

6J. Шарики

Ограничения задачи позволяют решать ее «в лоб». При этом получается хорошая задача на технику.

Будем искать длинную цепочку и удалять ее – сдвигать конец массива налево, уменьшая размер.

Как всегда, удобнее поставить барьерный элемент. В этом случае даже два барьерных элемента, например -1 и -2, чтобы не обрабатывать конец последовательности отдельно.

```
for (int i = 0; i < N; i++)
{
     // считаем количество одинаковых от i-го
```

```
     int k = 1;
     while (a[i] == a[i + k])
     {
          k++;
     }
     if (k >= 3)
     {// сдвигаем
          for (int j = i; j < N - k; j++)
          {
               a[j] = a[j + k];
          }
          N -= k; // теперь шариков на k меньше
          s += k; // удалили еще k шариков
          break;
     }
}
```

Таких действий нужно совершить максимум $\frac{N}{3}$ раз.

Обязательно хорошо разберитесь, как работает этот код. Особенно важна операция сдвига.

Занятие 6.Справочник

Создание массива

```
int[] arr = new int[10];
```

или

```
int arr[] = new int[10];
```

Создается массив с именем arr размером на 10 ячеек типа int.

С инициализацией:

```
double[] arr = new double[n]{2.0, 3.5, -1};
```

Ввод (считывание) массива из N элементов

```
for (int i = 0; i < N; i++)
{
     a[i] = in.nextInt();
}
```

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 IOEx4ϕception
   {
      in.nextToken();
      return (int)in.nval;
   }
}
```

Подробности чтения больших объемов данных смотрите в Справочнике к Занятию 5.

Вывод всех элементов массива

```
for (int i = 0; i < N; i++)
{
     out.print( a[i] + " ");
}
```

Поиск максимума

```
int max = a[0];
for (int i = 1; i < N; i++)
{
     if (max < a[i])
     {
          max = a[i];
}
```

Поиск индекса максимального

```
int imax = 0;
for (int i = 1; i < N; i++)
{
     if (a[imax] < a[i])
     {
          imax = i;
}
```

Поиск индекса заданного числа в массиве

```
int r = -1; // индекс элемента, равного x
int i = 0;
while (a[i] != x && i < N)
{
```

```
        if (a[i] == x) r = i;
        i++;
}
```

Если r после цикла останется -1 – элемент не найден.

Вывод массива в обратном порядке

1 вариант

```
for (int i = 0; i < N / 2; i++)
{
        int t = a[i];
        a[i] = a[N - i - 1];
        a[N – i - 1] = t;
}
```

2 вариант (с указателями)

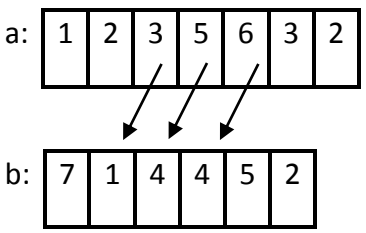
```
while(left < right)
{
        int t = a[left];
        a[left] = a[right];
        a[right] = t;
}
```

Копирование массивов

static void	**arraycopy**(Object src, int srcPos, Object dest, int destPos, int length) Copies an array from the specified source array, beginning at the specified position, to the specified position of the destination array.

Пример

```
int[] a = new int[]{1, 2, 3, 5, 6, 3, 2};
int[] b = new int[]{7, 1, 4, 4, 5, 2}
System.arraycopy(a, 2, b, 3, 1);
```



Из массива a, начиная со второго (третьего по счету) в массив b скопируются 3 символа. Причем, они будут размещены с первого (второго по счету) места в массиве b.

Таким образом, массив b станет таким: 7, 3, 5, 6, 5, 2