

# **Программирование на языках высокого уровня**

## **Тема 3. Сложные типы данных. 3.1. Массивы**

# Массивы

---

- **Массив – это именованный набор однотипных переменных, расположенных в памяти непосредственно друг за другом, доступ к которым осуществляется по индексу.**

# **Особенности массивов:**

---

- **все элементы имеют один тип**
- **весь массив имеет одно имя**
- **все элементы расположены в памяти рядом (следовательно, зная адрес первого элемента в памяти и индекс элемента, можно всегда однозначно найти указанный элемент массива)**

# Массивы

---

- Массивы различаются по следующим признакам:
  - по типу хранимых данных:  
массив целых чисел, массив символов, массив строк и т.д.
  - по количеству размерностей:  
одномерные, двумерные, трехмерные и т.д.

# Массивы

- Одномерные

$A =$

0	1	2	3

$A[2]$

- Двумерные

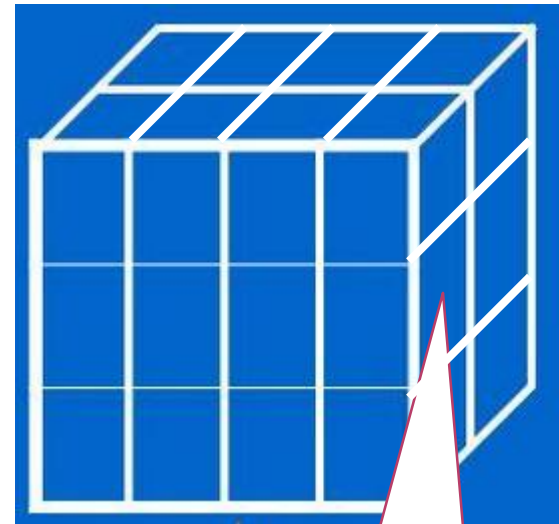
$B =$

	0	1	2	3
0				
1				
2				
3				
4				

$B[3][2]$

- Трехмерные

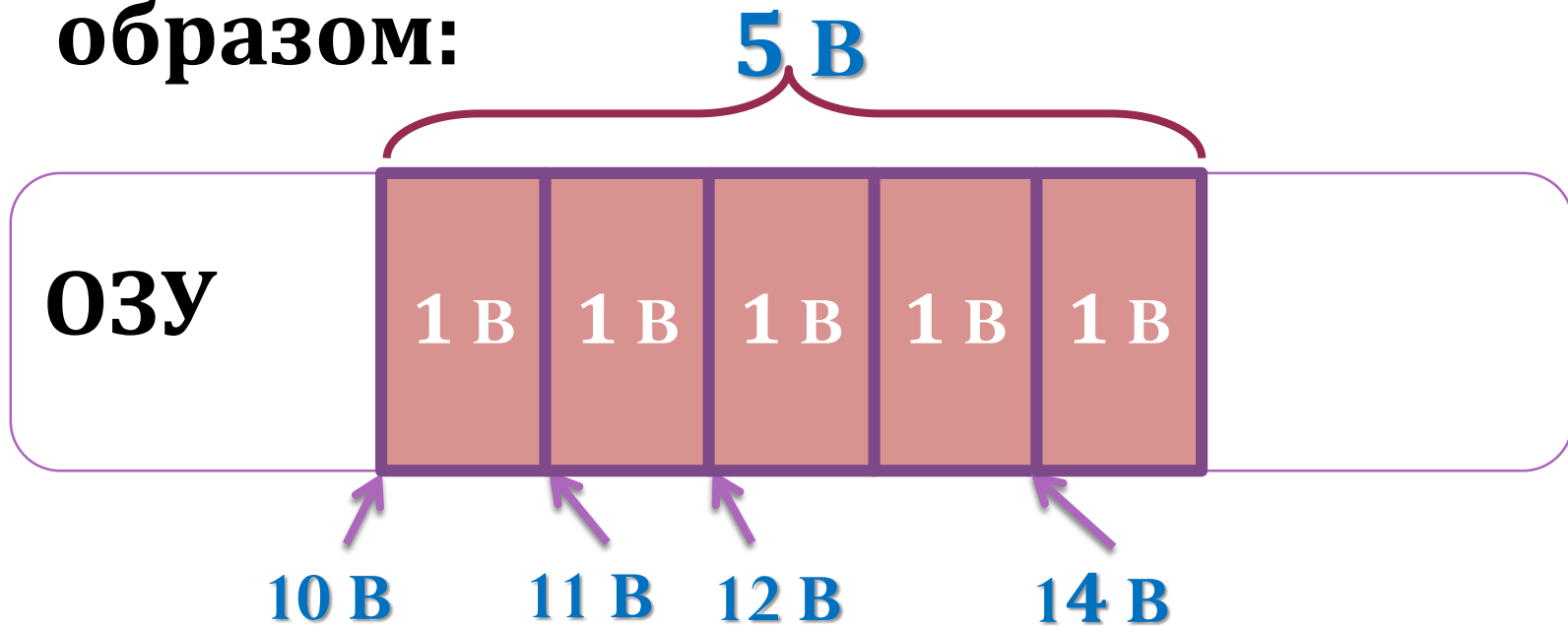
$C =$



$C[1][3][0]$

# Особенности хранения массивов

- Например, пусть имеется массив из 5 элементов типа byte.
- Тогда в памяти массив представляется следующим образом:



# Особенности хранения массивов

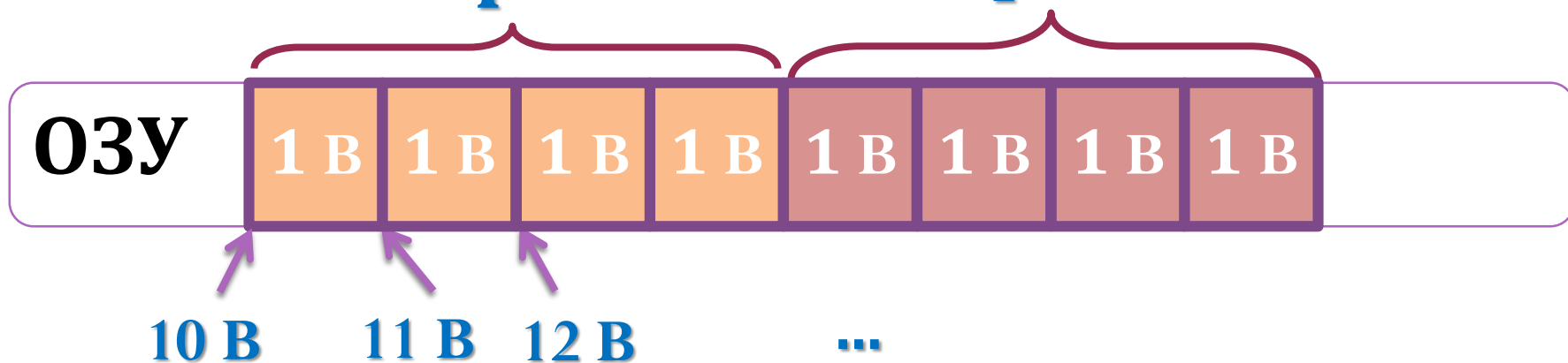
- Многомерные массивы также в памяти занимают линейное адресное пространство

$V =$

	0	1	2	3
0				
1				

0 строка

1 строка



# Массивы

---

- Массивы различаются по следующим признакам:
  - по способу выделения памяти:
    - *статические* массивы – память выделяется на этапе компиляции (Delphi)
    - *динамические* массивы – память выделяется во время выполнения (Delphi, Java, C, C#...)



# Динамические массивы

---

- Как правило, на этапе написания программы размер массива не известен.
- Если используются статические массивы, то нужно резервировать размер с «запасом» (неэкономное расходование памяти)
- *Динамические* массивы позволяют выделять ровно столько памяти, сколько требуется.

# **Динамические массивы**

---

- **Динамические массивы представляются в памяти ссылками, т.е. любая переменная - "динамический массив" является указателем на непрерывный участок динамической памяти.**

# Особенности использования динамических массивов

---

- При **объявлении** массива указывается только тип его элементов.

Java

**тип** [ ] имя\_массива;

**тип** имя\_массива [ ];

# Особенности использования динамических массивов

---

Java:

```
import ...  
public class Ex1{  
    psvm(String[] args){  
        ...  
        int [] a;  
        float b[];  
        ...  
    }  
}
```

# Особенности использования динамических массивов

---

- **Выделение памяти** под массив осуществляется в самой программе.

---

имя\_массива=**new** **тип**[*размер*];

**тип**[ ]имя\_массива = **new** **тип**[*размер*];

---

# Особенности использования динамических массивов

---

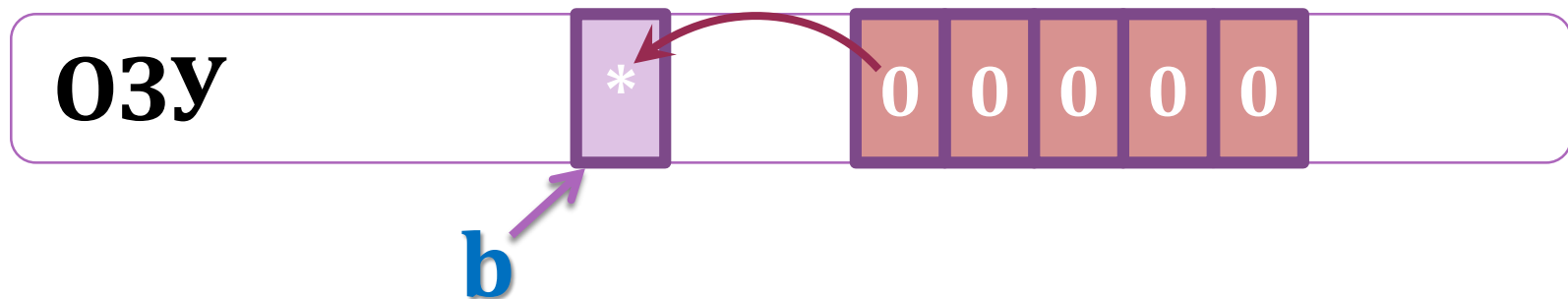
Java:

```
import ...  
public class Ex1{  
    psvm(String[] args){  
        Scanner sc = ...;  
        int n=sc.nextInt();  
        int [] a= new int[n];  
        ...  
    }  
}
```

# Динамические массивы

---

- После создания динамического массива адрес блока выделенной под него памяти записывается в переменную-массив (имя\_массива).



# Особенности использования динамических массивов

---

- **Длина** массива:

Java

*имя\_массива.length*

- **Индексы** только числовые,  
нумерация элементов с нуля!
- **Размер массива** в Java **изменять  
нельзя!**



# Особенности использования динамических массивов

---

- **Перебор элементов:**

- *Java:*

```
int []a = new int[10];  
for(int i=0; i<a.length; i++){  
    a[i] = 0;  
}
```

# Особенности использования динамических массивов

---

- Освобождение памяти по  
окончанию использования  
динамического массива:

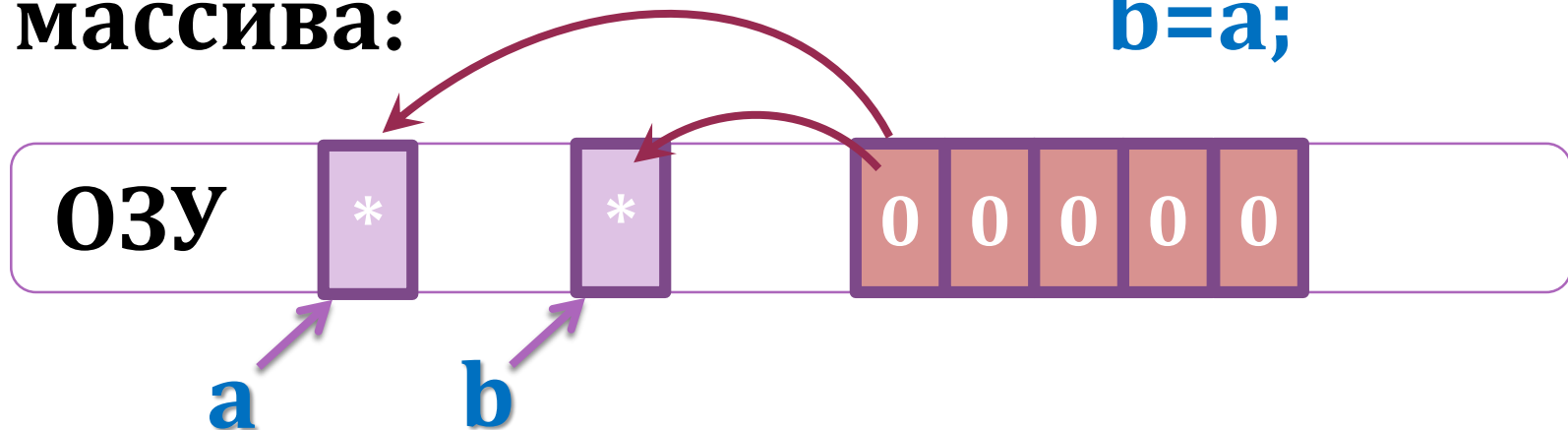
Java

**Не требуется!**  
(т.к. есть сборщик мусора)

# Особенности использования динамических массивов

- **Присваивание массивов:**
- Т.к. переменные - динамические массивы - это всего лишь адреса, то при присваивании динамических массивов копируется только адрес массива:

**b=a;**



# Многомерные динамические массивы

---

- *Объявление:*

```
тип [ ][ ]...[ ] имя_массива;
```

```
тип имя_массива [ ][ ]...[ ] ;
```

# Многомерные динамические массивы

---

- Выделение памяти
- Память должна выделяться под каждую размерность

```
массив=new тип[ размер1] [размер2]  
... [размерN ];
```

# Многомерные динамические массивы

---

Java:

```
...  
public static void main(String[] args){  
    Scanner sc = new Scanner(System.in);  
    int n=sc.nextInt();  
    int m=sc.nextInt();  
    int [][] a=new int[n][m];  
    ...  
}
```

# Многомерные динамические массивы

---

- Доступ к элементам

Java

*массив***[инд1][инд2]...[индN];**

# Способы перебора элементов массивов

---

Часто при работе с массивами задача ставится так, что требуется все элементы или их часть обработать одинаково. Для такой обработки организуется *перебор элементов*.



# Способы перебора элементов массивов

---

*Схему перебора* элементов массива можно охарактеризовать:

- направлением перебора;
- количеством одновременно обрабатываемых элементов;
- характером изменения индекса.

# Способы перебора элементов массивов

---

По *направлению перебора*  
различают схемы:

- от первого элемента к последнему;
- от последнего элемента к первому;
- от обоих концов к середине.

# Способы перебора элементов массивов

---

27

**В массиве одновременно можно обрабатывать один, два, три и т.д. элемента.**

**Часто в качестве параметра цикла используется индекс массива.**

**Обратите внимание также на то обстоятельство, что после изменения индекса его необходимо сразу же проверить на попадание в заданный диапазон, иначе возможны ошибки.**

# Общие правила организации перебора

---

**В правильно построенной схеме обязательно должны присутствовать:**

- **блок установки начальных значений индексов массива,**
- **блок проверки индекса,**
- **блок изменения индекса (для перехода к следующему элементу массива)**

- Перебор всех элементов массива по одному
- Перебор элементов массива с четными (нечетными) индексами
- Перебор элементов массива с индексами кратными  $k$
- Перебор всех элементов массива с концов к середине
- Перебор двух (трех, четырех и т.д.) соседних элементов
- И.т.д.

# Правило обработки массивов

---

- Для перебора элементов массива обычно применяют циклы с параметром.
- При этом, параметры цикла используются в качестве индексов элементов.
- Циклы запускаются отдельно по каждой размерности.
- Назначение каждого цикла определяется при обращении к элементу (напр.  $A[i][j]$  или  $A[j][i]$  и т.д.)

# Обработка массивов

---

# Обработка массивов

---

- **Инициализация массивов**
- **Заполнение (с клавиатуры и случайным образом) и распечатка элементов массива**
- **Поиск элементов в массиве**
- **Перестановка элементов**
- **Сортировка**
- **и др**



# Обработка массивов

---

## Инициализация массивов

**Инициализация массива – это заполнение массива начальными значениями при его создании.**

***По умолчанию, массив заполняется «нулевыми» значениями, предусмотренными в типе.***

# Обработка массивов

---

## Инициализация массивов в *Java*

*Явная инициализация:*

1. **тип[ ]** имя\_массива =  
**new** **тип[ ]** {v1, v2, ..., vN};
2. **тип[ ]** имя\_массива = {v1, v2, ..., vN};

**Количество элементов в массиве  
определяется по числу указанных  
значений.**

# Обработка массивов

---

- *Пример*

```
public static void main(String[] args) {  
    int[] nums={99,10,100,18,78,23,63, 49 };  
    int avg = 0;  
    for (int i = 0; i < nums.length; i++) {  
        avg += nums[i];  
    }  
    avg /= nums.length;  
    System.out.println("Среднее: " + avg);  
}
```

# Обработка массивов

---

## Инициализация многомерных массивов

```
тип [ ][ ] имя_массива = {  
    {v11, v12, ..., v1N},  
    {v21, v22, ..., v2N},  
    .... ,  
    {vM1, vM2, ..., vMN}  
};
```

# Обработка массивов

---

*Инициализация:*

```
int[ ][ ] s = { {1, 1, 3},  
                 {2, 4, 6},  
                 {3, 6, 8},  
                 {4, 1, 15}  
               };
```

# Обработка массивов

---

## Заполнение массива с клавиатуры:

Одномерный: `int []a = new int[n];`  
`for (int i=0; i<a.length;i++)`  
`a[i]=sc.nextInt();`

---

Двумерный:

```
int [][]a = new int[n][n];
for (int i=0; i<a.length;i++)
    for (int j=0; i<a[0].length;j++)
        a[i][j]=sc.nextInt();
```

# Работа со случайными числами в Java

---

1. **Math.random()** – возвращает псевдослучайное вещественное число из диапазона  $[0,1)$ .

- Примеры.

**a = Math.random();** // вещ.ч.  $[0,1)$

**b = (int) (Math.random()\*5);** //целое  $[0,4]$

**c = Math.random()\*5+3;** // вещ.ч.  $[3,8)$

**d = (int) (Math.random()\*11-5);**  
// целое  $[-5,5]$

# Работа со случайными числами в Java

---

2. Класс **Random** – (пакет **java.util**) содержит методы для генерации случайных чисел:
- Для использования требуется создать экземпляр класса одним из следующих конструкторов:
- ```
Random rnd = new Random();
```
- ```
Random rnd = new Random(long seed);
```



# Работа со случайными числами в Java

---

2. Класс **Random** – (пакет **java.util**) содержит методы для генерации случайных чисел:
- **nextInt()** – случ.значение типа **int**
  - **nextInt(int n)** – случ.знач. от 0 до n
  - **nextFloat()** – случ.значение из **float**
  - **nextDouble()** – случ.значение **double**
  - и др.

# Работа со случайными числами в Java

---

```
import java.util.Random;

public static void main (String[] args){
    Random rnd = new Random();
    int x = rnd.nextInt(); // x=1102829946
    int y = rnd.nextInt(100); // y=17
    float f = rnd.nextFloat(); // f=0.9788674
    double d = rnd.nextDouble();
    // d=0.08710495875238378
}
```

# Обработка массивов

---

## Заполнение массива случайным образом

```
Random rnd = new Random();  
int []a = new int[n];  
int []b = new int[n];  
for (int i=0; i<a.length;i++) {  
    a[i]=(int)(Math.random()*30-5);  
    b[i]=rnd.nextInt(30)-5;  
}
```

# Обработка массивов

---

## Распечатка массива:

Одномерный:

```
for (int i=0; i<a.length; i++)  
    System.out.printf("%d ", a[i]);
```

---

Двумерный:

```
int [][]b = new int[n][m];  
for (int i=0; i<b.length;i++) {  
    for (int j=0; j<b[i].length;j++) {  
        System.out.printf("%d ", b[i][j]);  
    }  
    System.out.println();  
}
```

# Поиск элемента в массиве

---

## Постановка задачи

- Пусть  $A = [a_1, a_2, \dots]$  – одномерный массив и  $b$  – некоторый элемент, обладающий некоторым свойством  $P$ .
- Найти место элемента  $b$  в массиве  $A$ .
- *Примеры:*
  - Найти позицию первого (второго и т.д.) элемента кратного 7
  - Определить позицию максимального элемента массива

# Поиск элемента в массиве

---

Наиболее простые и часто оптимальные алгоритмы основаны на последовательном просмотре массива ***A*** с проверкой свойства ***P*** на каждом элементе.

# Поиск максимального элемента массива

---

- *Алгоритм:*

1. В **max** записываем первый элемент массива
2. Запускаем цикл по всему массиву
  - Если элемент массива больше, чем **max**, то сохранить его (элемент) в переменную **max**
3. Вывести значение **max**

# Поиск максимального элемента массива

---

```
public static void main(String[] args){  
    int[] x;  
    int i,n, max;  
    ...  
    //Заполнение массива  
    ...  
    max = x[0]; {считаем, что первый элемент  
                  максимальный}  
    for (int i=0 ; i<n; i++){  
        if (x[i] > max) max = x[i];}  
    souf ("max = %d", max);
```



# Поиск максимального элемента массива и его позиции

---

- *Алгоритм:*

1. Сохраняем в переменную **imax** индекс первого элемента массива
2. Запускаем цикл по всему массиву
  - Если элемент массива больше, чем элемент, расположенный на месте **imax**, то сохранить его номер (элемента) в переменную **imax**
3. Вывести значение элемента под номеров **imax**

# Поиск позиции максимального элемента

---

```
public static void main(String[] args){  
    int[] x;  
    int i,kol;  
    ...  
    int imax = 0; /*считаем, что  
                    максимальный на первом месте*/  
    for (int i=1; i< n ; i++){  
        if (x[i] > x[imax]) imax =i;  
        //сохраняем новую позицию  
        souf ("max = %d  \n Позиция = %d",  
            x[imax], imax);  
    }  
}
```

# **Поиск позиции максимального элемента**

---

## **Самостоятельно...**

- **Дан массив, заполненный случайными целыми числами в диапазоне  $[-25 ; 30]$ .**

**Определить значение и позицию минимального элемент массива, расположенного после первого элемента равного 10.**

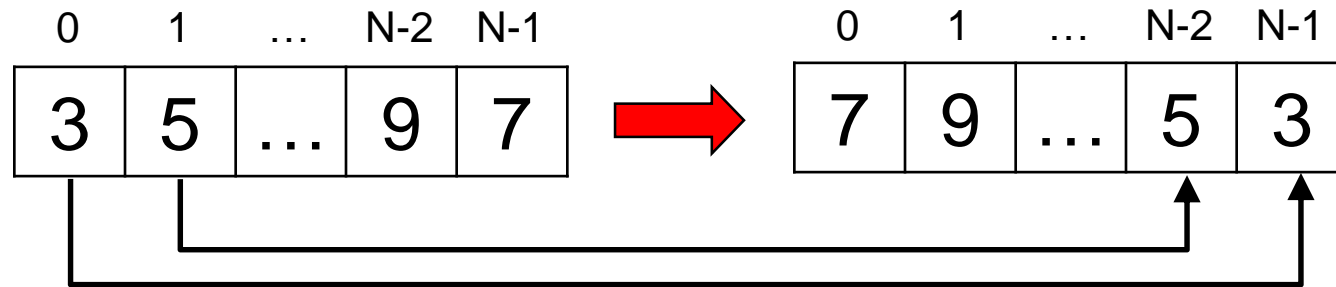
# Обработка массивов

---

- **Перестановка элементов**
- **Сдвиг элементов**
- **Сортировка массива**
- **и т.д.**

# Реверс массива

**Задача:** переставить элементы массива в обратном порядке.



**Алгоритм:**

поменять местами  $A[0]$  и  $A[N-1]$ ,  
 $A[1]$  и  $A[N-2]$ , ...

сумма индексов  $N-1$

# Реверс массива

---

## Псевдокод:

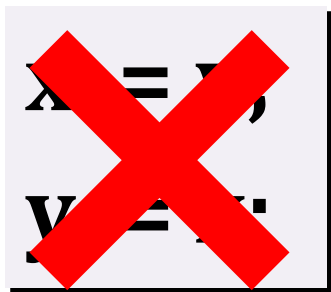
```
for (int i=0; i<n/2 ; i++)  
    { поменять местами  
      a[i] и a[n-i-1] }
```



Что неверно?

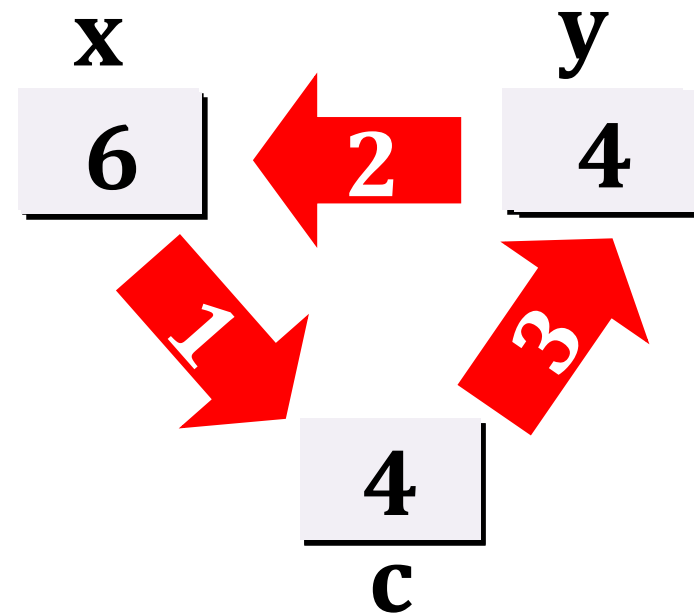
# Как переставить элементы?

- *Перестановка элементов*
- **Задача:** поменять местами содержимое двух ячеек памяти.



~~`x = y;  
y = x;`~~

```
c = x;  
x = y;  
y = c;
```



# Программа

```
int n = 10;  
int A[]=new int[n];
```

{ заполнить и вывести массив }

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

{ вывести полученный массив }



# Сортировка массива

---

## **Суть сортировки:**

- **Последовательно просматривается массив и сравнивается каждая пара элементов между собой.**
- **При этом "неправильное" расположение элементов устраняется путем их перестановки.**
- **Процесс просмотра и сравнения элементов повторяется до тех пор, пока массив не будет отсортирован.**

# Сортировка массива по возрастанию

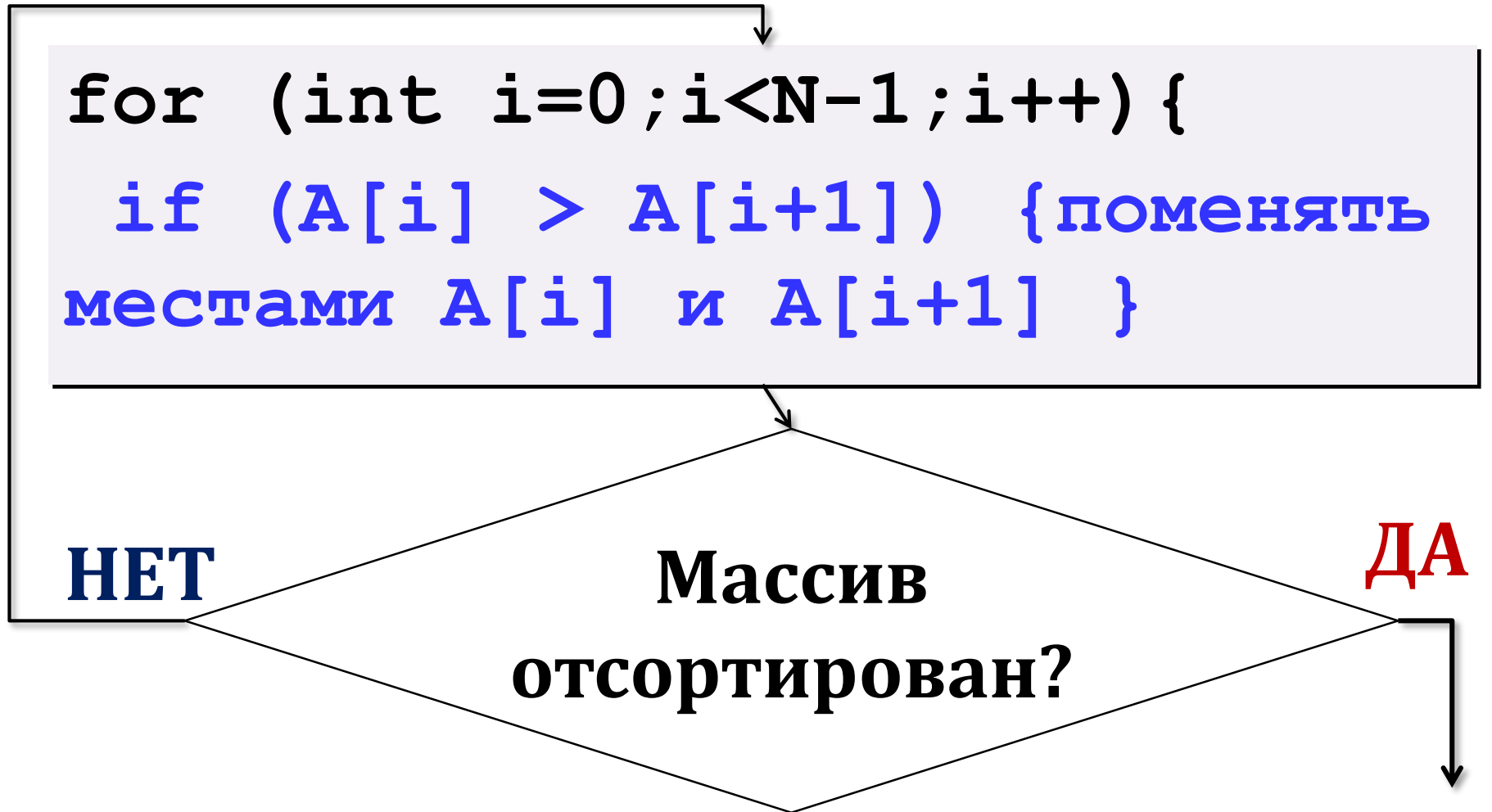
---

```
for (int i=0; i<N-1; i++) {  
    if (A[i] > A[i+1]) { поменять  
местами A[i] и A[i+1] }
```

**НЕТ**

**Массив  
отсортирован?**

**ДА**



# Сортировка массива по возрастанию

```
int i, n;    f : boolean;  a:mas;  
boolean isSorted;
```

```
...  
do {  
1   isSorted = true;  
   for (i=0; i<n-1;i++)  
2   if (a[i] > a[i+1]){  
3       int temp = a[i];  
       a[i] = a[i+1];  
       a[i+1] = temp;  
4   isSorted = false;  
   }  
while (isSorted);
```

1. Считаем, что массив отсортирован
2. Проверка расположения элементов
3. Перестановка элементов в порядке возрастания
4. Массив неотсортирован

# Решение задач на одномерные массивы

---

# 1. Найти среднее арифметическое кратных 5 элементов

---

```
...
int k=0;
double s=0;
for (int i=0; i< a.length; i++){
    if (a[i] % 5 == 0) {
        s += a[i];
        k++;
    }
if (k > 0) soutf("ср.арифм. =%.3f", s/k);
sout("нет кр.5 элементов");
```

## 2. Найти количество простых элементов

---

```
...k=0;
for (int i=0; i<a.length; i++){
    elemIsSimple = true;
    for (int j=1; j<a[i]/2; j++){
        if (a[i] % j == 0) {
            elemIsSimple = false;
            break;
        }
    }
    if (elemIsSimple) k++;
}
sout("Кол-во простых = "+ k);
```

### 3. Определить, является ли максим. элемент - простым числом

---

...

```
int max = a[0];  
for (int i=1; i<a.length;i++) {  
    if (a[i] > max) max = a[i];
```

```
boolean el_Simple = true;  
for (int j=2;j<max/2&&el_Simple; j++)  
    if (max % j == 0) el_Simple = false;
```

```
if (el_Simple) sout("Макс. Простой"); else  
sout("Макс. – составной");
```

## 4. Найти сумму делителей минимального четного элемента

*{1.4. Найти сумму делителей  
минимального элемента}*

Java:

```
int imin = -1;  
for (int i=0; i <a.length; i++)  
    if (a[i]%2==0&&  
        (imin<0||a[i] < a[imin])) imin =i;
```

```
int sumDelit = 0;  
for (int j=2; j<a[imin]/2; j++)  
    if (a[imin] % j == 0) sumDelit +=j;
```

```
sout("Сумма делит мин. элем = "+  
    sumDelit);
```



## 5. Вывести положительные элементы массива

---

```
...  
positiveExists = false;  
for (int i=0; i<n; i++){  
    if (a[i] > 0) {  
        sout("%4d", a[i]);  
        positiveExists = true;  
    }  
if (!positiveExists)  
    sout("Положительных нет");
```

# Решение задач

---

6. *Определить произведение чисел, стоящих на нечетных местах, расположенных после второго кратного 5 элемента. Если таких элементов нет, вывести соответствующее сообщение*
  1. Шаг1 – найти позицию второго элемента кратного 5
  2. Шаг2 – вычислить произведение, начиная с найденной выше позиции и до конца массива

*6. Найти произведение чисел,  
стоящих на нечетных местах,  
{ расположенных после второго  
кратного 5 элемента*

```
int kr_5 =0;
for (int i=0; i<a.length;i++){
    if (a[i] % 5 == 0) {
        kr_5++;
        if (kr_5 == 2) {
            pos_kr5 =i;
            break; }
    }
}
```

*6. Найти произведение чисел,  
стоящих на нечетных местах,  
{ расположенных после второго  
кратного 5 элемента*

```
int proiz = 1;  
if (kr_5 == 2) {  
    for (int j=pos_kr5+1; i<a.length; i++)  
        if (j % 2 == 0) proiz *= a[j];  
    souf("Произведение = %d", proiz);  
}  
else  
    sout("Нет двух кратных 5!");
```

# Решение задач на матрицы

---

# Решение задач на матрицы

Дана **матрица** размерностью  $N \times M$ .

Требуется:

1. Заполнить матрицу следующим образом:

1	2	3	4	5	...
11	12	13	14	15	...
21	22	23	24	25	...
...	...	...			

```
for (i=0; i<n; i++)  
for (j=0; j<m; j++)  
    a[i][j]=i*10+j+1;
```

1	1	1	1	1	1
1	2	2	2	2	2
1	2	3	3	3	3
1	2	3	4	4	4
...	...	...			

```
for (i=0; i<n; i++)  
for (j=0; j<m; j++)  
    if (j>=i) a[i][j]=i+1  
    else a[i][j] = j+1;
```

## 2. Найти максимальный и минимальный элементы :

---

```
int min = a[0][0], max = a[0][0];
for (i=0; i<n; i++) {
    for (j=0; j<m; j++) {
        if (a[i][j] < min) min = a[i][j];
        if (a[i][j] > max) max = a[i][j];
    }
}
printf("min=%d \t max = %d", min, max);
```

### 3. Определить номер строки, в котором расположен максимальный элемент:

```
int imax = 0, jmax = 0;
for (int i=0; i< n; i++){
    for (int j = 0; j< m; j++){
        if (a[i][j] > a[imax][jmax]){
            imax = i;
            jmax = j;
        }
    }
    sout("Номер строки с макс. эл." +
        imax);
```



#### 4. Найти максимальный и минимальный элементы в каждой строке:

```
for (int i=0; i<n; i++){  
    int min = a[i][0], max = a[i][0];  
    for (int j=0; j<m; j++){  
        if (a[i][j] < min) min = a[i][j];  
        if (a[i][j] > max) max = a[i][j];  
    }  
    printf("Стр № %d min=%d  
        max=%d\n", i, min, max);  
}
```

## 5. Определить номер строки с минимальной суммой элементов:

```
for (int i=0; i<n; i++){  
    s = 0;  
    for (int j = 0; j< m; j++) {  
        s += a[i][j];  
    }  
    if (i==0 || s<min) {  
        min=s;  
        imin=i;  
    } }  
sout("№ строки с мин.суммой."+ imin);
```

**6. Получить массив  $X[N]$ ,  $i$ -тый элемент которого равен количеству кратных 5 элементов  $i$ -той строки матрицы:**

```
for (int i= 0; i<n; i++) {  
    x[i] = 0;  
    for (int j =0; j <m; j++)  
        if (a[i][j] % 5 == 0) x[i]++;  
}
```

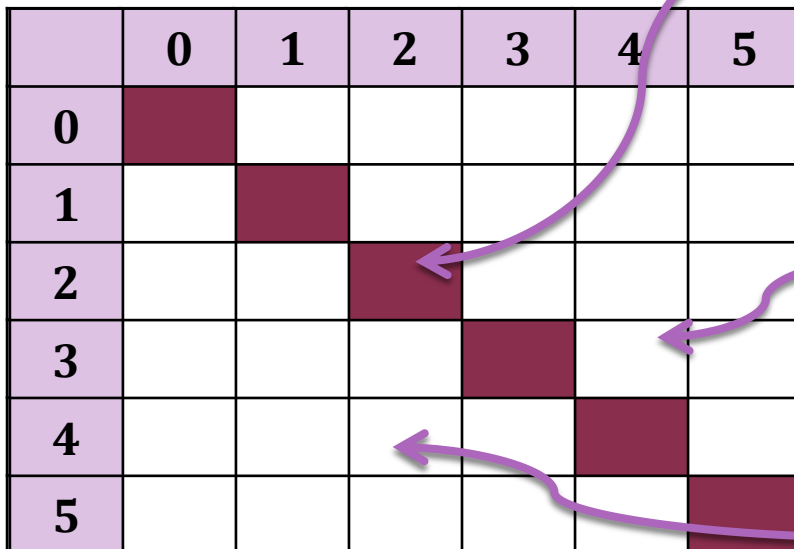
```
for (int i=0; i<n; i++)  
    souf("%4d", x[i]);
```

## Задача 2

- Дана квадратная матрица порядка  $N$ :

Пусть  $N = 6$

**Главная диагональ**



	0	1	2	3	4	5
0						
1						
2						
3						
4						
5						

Элементы на главной диагонали  
 $i = j$

Элементы выше главной диагонали  
 $i < j$

ниже главной диагонали  
 $i > j$

# Решение задач на матрицы

Дана квадратная матрица порядка  $N$  :

Пусть  $N = 6$

**Побочная диагональ**

	0	1	2	3	4	5
0						
1						
2						
3						
4						
5						

Элементы на побочной диагонали

$$i+j = n-1$$

Элементы выше побочной диагонали  $i+j < n-1$

ниже побочной диагонали  $i+j > n-1$

# Решение задач на матрицы

---

1. Требуется определить сумму элементов, расположенных выше главной диагонали и ниже побочной

	0	1	2	3	4	5
0						
1						
2						
3						
4						
5						

**S = 0;**

```
for (int i=0; i<N; i++){  
  for (int j=0; i<N; j++){  
    if (i<j && i+j>N-1)  
      S+=a[i][j];  
  }  
}
```

# Решение задач на матрицы

**2. Вычесть минимальный элемент главной диагонали из всех четных элементов матрицы**

	0	1	2	3	4
0					
1					
2					
3					
4					

```
int min = a[0][0];
```

```
for (int i=1; i<N; i++)
```

```
    if (a[i][i]<min) min = a[i][i];
```

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

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

```
        if (a[i][j] % 2 == 0) a[i][j]-= min;
```

## 7. Отсортировать строки матрицы по возрастанию суммы четных элементов

---

*7.1 Вычислить суммы четных  
элементов строк и записать их в  
одномерный массив  $s$*

```
for (int i = 0; i < n; i++) {  
    s[i] = 0;  
    for (int j = 0; j < m; j++)  
        if (a[i][j] % 2 == 0)  
            s[i] += a[i][j];  
}
```



## 7.2 Отсортировать массив $s$ . При сортировке переставлять элементы строк

```
do{  
    f = true;  
    for (int i=0; i<n-1; i++)  
        if (s[i]>s[i+1]) {  
            t=s[i]; s[i]=s[i+1]; s[i+1] = t;  
            for (int j = 0; j< m; j++){  
                t=a[i][j]; a[i][j]=a[i+1][j];  
                a[i+1][j] = t;  
            }  
            f= false;  
        }  
}while (f);
```

Перестановка  
элементов массива  
 $s$  по возрастанию

Перестановка строк  
матрицы  $a$  по возрастанию  
суммы элементов

---

# **Массивы как аргументы методов и как тип результата**

# Передача массивов в методы

---

- **тип** **имя**( **тип\_мас** [ ] **имя\_мас**) { }

*Пример*

```
static int max (int[ ] a) {  
    int res = a[0];  
    for (int i=1; i<a.length; i++){  
        if (res<a[i]) {res=a[i];}    }  
    return res;  
}
```

# Передача массивов в методы

---

- **Функция возвращает массив**

**тип** [ ] **имя**( **тип\_мас** [ ] **имя\_мас** ) { }

# Передача массивов в методы

---

## *Пример*

```
static int[] maxInStrings(int[ ][ ] a){  
    int[] c = new int[a.length];  
    for (int i=0;i<a.length; i++){  
        c[i] = a[i][0];  
        for (int j=1; j<a[i].length;j++){  
            if (c[i]<a[i][j]) c[i]=a[i][j];  
        }  
    }  
    return c; }  
}
```

---

# Оператор цикла `foreach` в Java

# Оператор цикла foreach

---

- В языке Java определен еще один вариант оператора цикла **for**, который также называют **foreach**.
- Он служит для циклического обращения к элементам коллекции, представляющей собой группу объектов.
- Массивы также являются коллекциями!

# Оператор цикла foreach

---

- Общая форма оператора:

```
for (тип имя_перемен : коллекция) {  
    // тело цикла  
}
```

В таком цикле **имя\_перемен**  
поочередно будет получать  
значение следующего элемента  
**коллекции**



# Оператор цикла foreach

---

- Пример:

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

```
...
```

```
for (int x : a) {  
    System.out.printf("%3d ", x);  
}
```

# Оператор цикла `foreach`

---

- При просмотре многомерного массива следует помнить, что цикл получает доступ к элементам массива, которые в свою очередь тоже массивы.
- Поэтому, для обработки многомерных массивов, необходимо организовывать вложенные циклы `foreach`.

# Оператор цикла foreach

---

- Пример:

```
int[ ][ ] b = new int[5][6];
```

```
...
```

```
for (int[ ] x : b) {
```

```
    for (int y : x) {
```

```
        System.out.printf("%3d ", y);
```

```
    }
```

```
}
```

# Оператор цикла foreach

---

```
for (тип имя_перем : коллекция) {  
    // тело цикла  
}
```

Следует иметь в виду, что  
*переменная цикла* в операторе  
foreach служит только для  
чтения!!!

Нет возможности обратиться к  
индексу!!!

# Оператор цикла foreach

---

- Пример:

```
int[ ][ ] b = new int[5][6];  
for (int[ ] x : b) {  
    for (int y : x) {  
        y = (int) (Math.random()*100);  
        System.out.printf("%3d ", y);  
    }  
}
```

Исходный массив *b* остается  
без изменений!!!

# Оператор цикла foreach

---

- Оператор **foreach** допускает циклическое обращение к массиву только в определенном порядке: от начала и до конца.
- Допустимо использование **break**;
- Оператор цикла **foreach** оказывается особенно полезным для работы с разными типами коллекций.

---

# Ступенчатые массивы

# Ступенчатые массивы

---

- **Ступенчатые массивы – это многомерные динамические массивы, в которых каждый массив следующего уровня может иметь разный размер.**



# Ступенчатые массивы

---

- Например, двумерный массив необязательно квадратный или прямоугольный.

1	5	3	8	6	7	9
4	1	0				
6	8	1	6	3	2	
7	9					
7	5	5	3	4		

# **Ступенчатые массивы**

---

- **Для задания ступенчатого массива необходимо выделять память только под первую размерность, а под остальные – отдельно по мере необходимости**

# Ступенчатые массивы

---

```
int [ ][ ] a = new int [3][ ];  
    a[0] = new int[5];  
    a[1] = new int[3];  
    a[2] = new int[7];
```

# Ступенчатые массивы

---

**Пример.**

**Требуется описать матрицу, число столбцов которой в строках случайное число от 1 до 5**

# Ступенчатые массивы:

## Пример на Java

---

```
Random rnd = new Random();  
int [ ][ ] x = new int[5][ ];  
for (int i=0; i<x.length; i++){  
    x[i] = new int [ rnd.nextInt(5)+1 ];  
    for (int j=0; j<x[i].length; j++){  
        x[i][j] = rnd.nextInt(20)-5;  
    }  
}
```

# Ступенчатые массивы

---

## Инициализация ступенчатого массива в Java

```
int [ ][ ] a = new int [ ][ ] {  
    {1, 1, 1, 1},  
    {2, 2, 2},  
    {1, 2, 3, 4, 5, 6, 9}  
};
```

Размерность массива определяется по набору значений

# Ступенчатые массивы

---

**На практике редко встречаются задачи, в которых нужно использовать ступенчатые массивы**