

实验报告：

1. 以下表达式是否能够通过编译？如果能够通过，表达式的类型是什么？表达式的值是什么？请尝试解释原因。

```
1 == True
1 == true
0 == false
2 + "ab"
2.3 + "ab"
2 + 'a'
2 * "ab"
2 * 'a'
1 + 1.0
1/3
1.0/3
```

1.1

因为 java 是语法严格的语言我们写错一个 keyword 时编译不会通过的；

1.2

在 java 语言数据类型要严格区分在这里 `1==true` `1` 是整形类型而 `true` 是 `boolean` 类型字面变量所以编译不会通过；

1.3

`0==false` 也是跟上面一样的例子整形变量和 `boolean` 字面变量不匹配的会报错；

```
System.out.println(1==true);
System.out.println(0==true);
```

1.4

`2 + "ab"` 如果我们把结果给我一个整形类型会把报错 因为整形类型不会接受字符串类型 如果是 `int` 型的包装类其实可以把字符串转换对应的数字但是在上面的例子是不行的 因为 `Integer.parseInt` 只能接受 0 到 9 的数字； 如果用 `Integer.parseInt` 这个方法先会检查是否是字面字符如果是会捕捉：

```
int n = Integer.parseInt(s: 2 + "ab");
System.out.println(n);
```

```
}
```

```
Exception in thread "main" java.lang.NumberFormatException: Create breakpoint : For input string: "2ab"
    at java.base/java.lang.NumberFormatException.forInputString(NumberFormatException.java:67)
    at java.base/java.lang.Integer.parseInt(Integer.java:662)
    at java.base/java.lang.Integer.parseInt(Integer.java:778)
    at MaxOf.main(MaxOf.java:7)
Disconnected from the target VM, address: '127.0.0.1:53646', transport: 'socket'
```

因为 `parseInt` 的方法是如下：

```
@Contract(pure = true)
public static int parseInt(String s) throws NumberFormatException {
    return parseInt(s, radix: 10);
}
```

所以当遇到字面字符会捕捉：

如果把结果给一个字符串型那会把前面的整形编译的时候自动转换字符串再合并他们呢
因为+在字符串操作时会合并两个字符串的。

2.5+ “ab” 也是一样的；

1.5

2+ 'a'

我们知道根据 `unicode` `a` 的十进制 `code` 等于 97 那么上面的表达式意味着 $97+2 = 99$
所以如果把结果给一个 `char` 型会输出 `c` 字符；

```
System.out.println(2+'a');
```

否则会输出对应 `unicode` 十进制：

```
connected to the t
99
Disconnected from
```

1.6

1.7

2* "ab" 编译不会通过因为 * 数学操作符不能对字符串进行操作;

```
System.out.println(2*"ab");
```

Operator '*' cannot be applied to 'int', 'java.lang.String'

1.8

2* 'a' 像 2+ 'a' 一样的 a 的代码等于 97 再程 2 = 194

```
System.out.println(2*'a');
```

1.9

1+1.0 在编程语言有一个类型升级的概念 每当我们把类型小的数据加减乘除一个比自己大的数据类型是会自动的转换那个大的类型 再上面的例子 1 是整形 1.0 是 double 所以结果是 2.0 类型为 double;

1.10

1/3 输出的结果可能不是我们希望得到的一个结果 因为两个数字都是整形所以结果也会是整形 则答案是 0

```
Connected to the target VM, a
0
Disconnected from the target
```

1.11

1.0/3

在这个例子 1.0 是 double 所以结果也是一个 double 则等于:

```
connected to the target VM, a  
0.3333333333333333  
Disconnected from the target
```

虽然 3 是整形但是因为一个是 double 又因为 int 小于 double 所以结果也是为 double;

2

2. 假设 `int a = 2147483647;` (即, `Integer.MAX_VALUE`). 请问以下语句的输出值是什么? 并解释原因.

```
System.out.println(a);  
System.out.println(a + 1);  
System.out.println(2 - a);  
System.out.println(-2 - a);  
System.out.println(2 * a);  
System.out.println(4 * a);
```

2.1

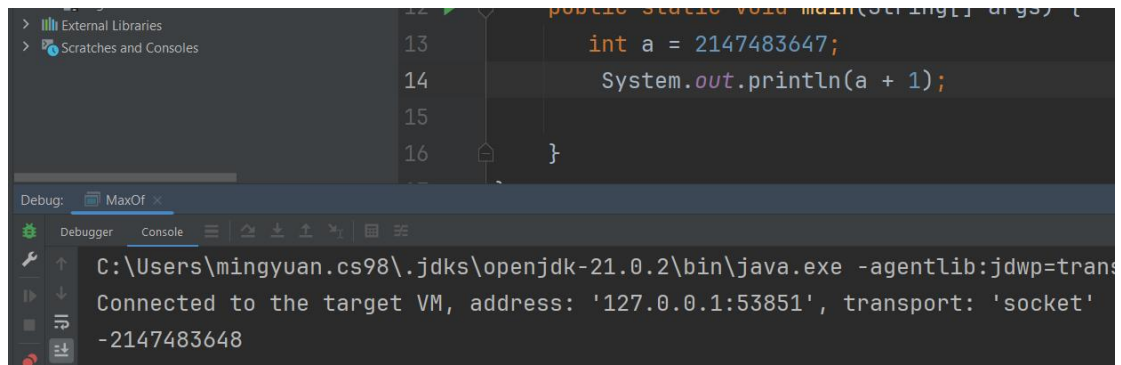
```
12 public static void main(String[] args) {  
13     int a = 2147483647;  
14     System.out.println(a);  
}
```

Debugger Console

```
C:\Users\mingyuan.cs98\jdk\openjdk-21.0.2\bin\java.exe -agentlib:jdwp=transport:dt_socket,address=127.0.0.1:53828,transport=socket  
Connected to the target VM, address: '127.0.0.1:53828', transport: 'socket'  
2147483647
```

第一个会输出准确数字因为 int 的最大数是 2147483647 所以可以容纳这么大的一个数,

2.2



```
12 public static void main(String[] args) {
13     int a = 2147483647;
14     System.out.println(a + 1);
15 }
16
```

Debug: MaxOf x

Debugger Console

C:\Users\mingyuan.cs98\jdk\openjdk-21.0.2\bin\java.exe -agentlib:jdwp=transport=dt_socket,address=127.0.0.1:53851,transport=socket

Connected to the target VM, address: '127.0.0.1:53851', transport: 'socket'

-2147483648

这个例子也很合理因为 int 最大值二进制表示法是：01111111111111111111111111111111

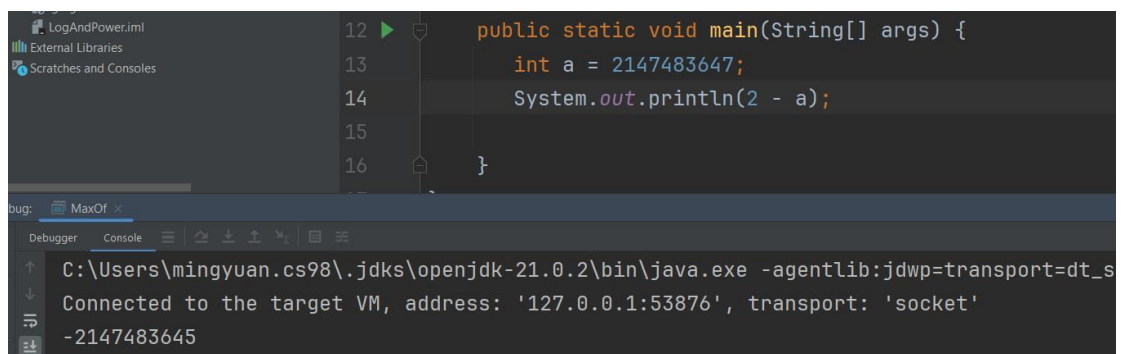
如果我们在加已会溢出所以会跳到 int 的另一个极端：

数字会变为：10000000000000000000000000000000

在表示二进制法是最高位也就是说第一位是该数字的符号 整形的话是 0 负数的话是 1

所以会输出 int 的最大负数；

2.3



```
12 public static void main(String[] args) {
13     int a = 2147483647;
14     System.out.println(2 - a);
15 }
16
```

Debug: MaxOf x

Debugger Console

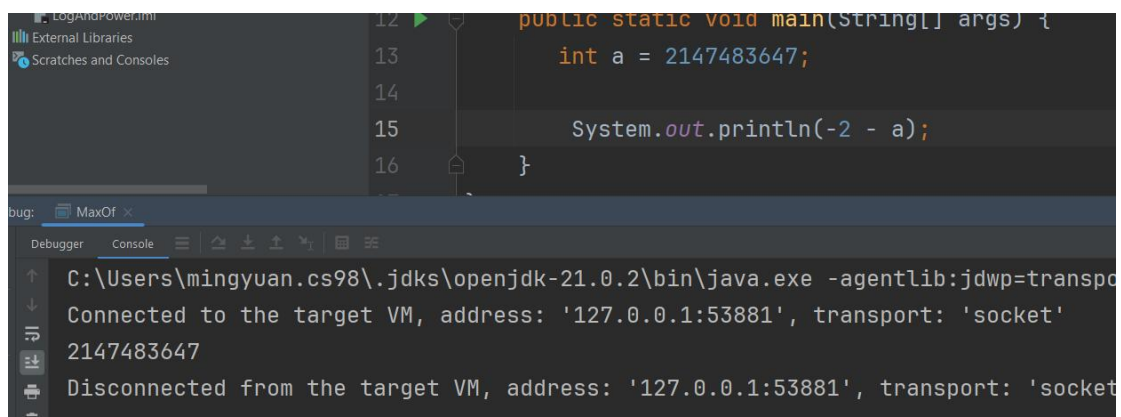
C:\Users\mingyuan.cs98\jdk\openjdk-21.0.2\bin\java.exe -agentlib:jdwp=transport=dt_socket,address=127.0.0.1:53876,transport=socket

Connected to the target VM, address: '127.0.0.1:53876', transport: 'socket'

-2147483645

因为-2147483645 是在整形负数的范围内所以我们可以得到准确数字；

2.4



```
12 public static void main(String[] args) {
13     int a = 2147483647;
14     System.out.println(-2 - a);
15 }
16
```

Debug: MaxOf x

Debugger Console

C:\Users\mingyuan.cs98\jdk\openjdk-21.0.2\bin\java.exe -agentlib:jdwp=transport=dt_socket,address=127.0.0.1:53881,transport=socket

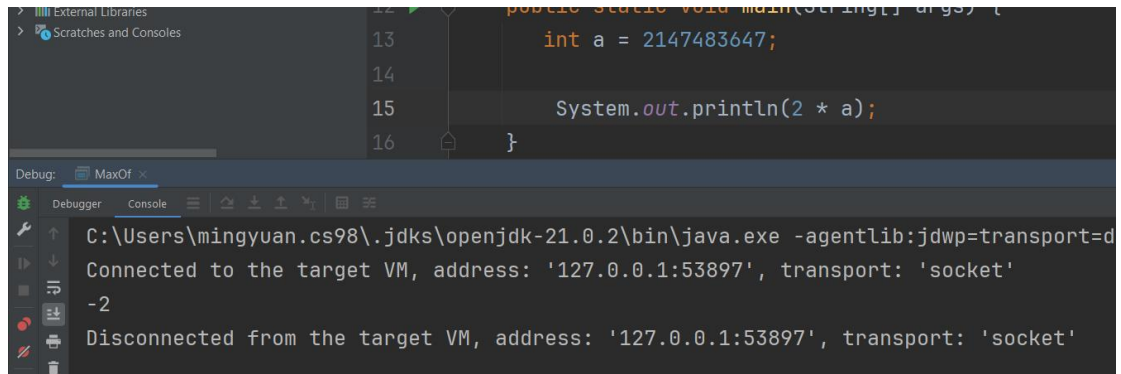
Connected to the target VM, address: '127.0.0.1:53881', transport: 'socket'

2147483647

Disconnected from the target VM, address: '127.0.0.1:53881', transport: 'socket'

因为 -2147483649 是小于 int 能够表示的最小数所以会导致溢出，则跑到 int 的另一端然后输出 int 整形能够表示的最大数；

2.5



```
public static void main(String[] args) {  
    int a = 2147483647;  
    System.out.println(2 * a);  
}
```

Debug: MaxOf x

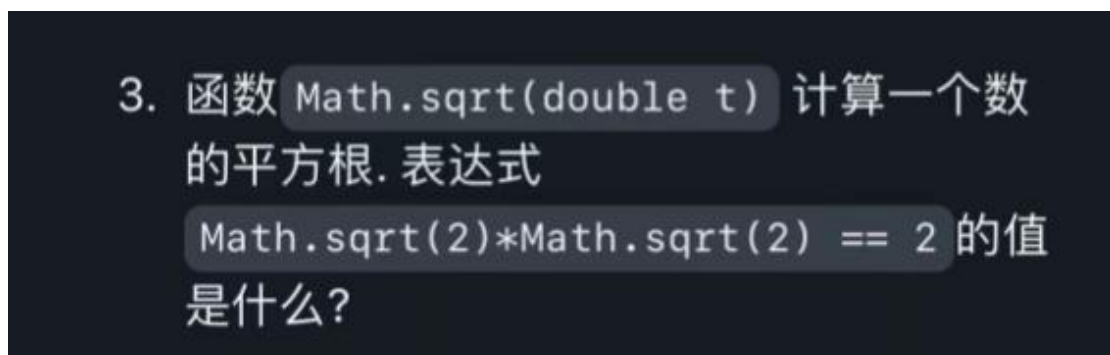
Debugger Console

C:\Users\mingyuan.cs98\.jdk\openjdk-21.0.2\bin\java.exe -agentlib:jdwp=transport=d
Connected to the target VM, address: '127.0.0.1:53897', transport: 'socket'
-2
Disconnected from the target VM, address: '127.0.0.1:53897', transport: 'socket'

因为 $2 * \text{int}$ 整形最大数所以它会跳到整形的另一端，而且数字 $*2$ 是左移一位所以
 $2 * 01111111111111111111111111111111 = 1111111111111111111111111111110 = -2$

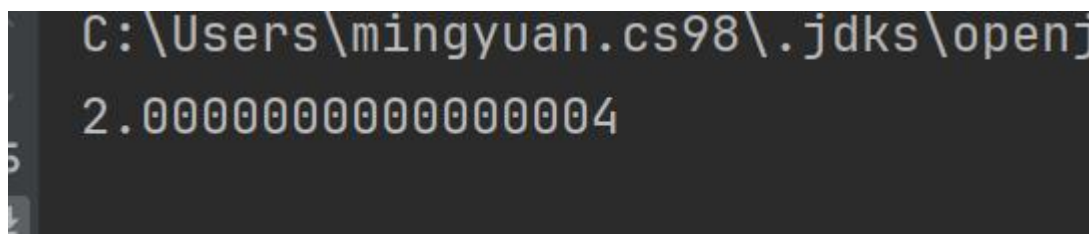
最后一个 $*4$ 也是一样的左移 2 位
 $4 * 01111111111111111111111111111111 = 1111111111111111111111111111100 = -4$

3

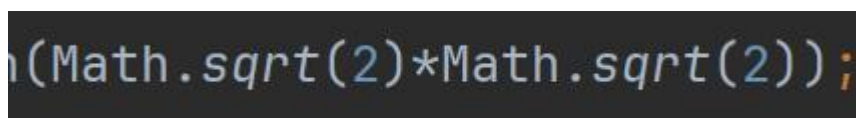


3. 函数 `Math.sqrt(double t)` 计算一个数的平方根. 表达式
`Math.sqrt(2)*Math.sqrt(2) == 2` 的值是什么?

在编程语言一个浮点数不能够完美的表示，因为数据类型像 `float` 和 `double` 分别能够比较准确的小点数是 6 位和 15 位 所输出的结果不会是 2.0 则会输出 `false`
我们来测试这个：



```
C:\Users\mingyuan.cs98\.jdk\openj  
2.000000000000000004
```



```
Math.sqrt(2)*Math.sqrt(2);
```

果然这个表达式不等于 2 所以他们是不等价的；

4

4. 给定命令行参数 x_1, y_1, x_2, y_2 . 计算平面上的点 (x_1, y_1) 和 (x_2, y_2) 的距离.

```
double x1,x2,y1,y2,diff,distance;

x1 = Double.parseDouble(args[0]); // 因为args里面的元素是字符串所以要转换double
y1 = Double.parseDouble(args[1]);
x2 = Double.parseDouble(args[2]);
y2 = Double.parseDouble(args[3]);

diff = Math.pow((x2-x1),2)-Math.pow((y2-y1),2);
distance = Math.sqrt(diff);
System.out.println("Distance is: "+distance);
```

输出:

```
D:\2024nian\java\lab2>java DistanceOfPoints 0 0 6 8
Distance is: 10.0

D:\2024nian\java\lab2>
```


5. 计算函数 $\log x, x, x \log x, x^2, x^3$ 在 $x = 1, 2, 4, 8, 16, \dots, 2048$ 处的值. 并比较它们的增长速度.

根据输出的结果我们可以的判断如果从小到大排序则: $\log x, x, x \log x, x^2, x^3$;

```
x^3: 8.50479330E9
X: 2047
Logx: 7.624130585661289
xLogx: 15606.595308848659
x^2: 4190209.0
x^3: 8.577357823E9
X: 2048
Logx: 7.6246189861593985
xLogx: 15615.219683654448
x^2: 4194304.0
x^3: 8.589934592E9
```

```
public class LogAndPower {

    public static void main(String[] args){

        for(int x = 1;x<2049;x++){
            double logOfX = Math.log(x);

            System.out.println("X: "+x);
            System.out.println("Logx: "+logOfX);
            System.out.println("xLogx: "+x*logOfX);
            System.out.println("x^2: "+Math.pow(x,2));
            System.out.println("x^3: "+Math.pow(x,3));

        }

    }
}
```


7

7. 给定5个整数 (通过命令行参数), 找出它们的中位数 (即第3大的数).

考虑到代码有点长的问题就没截图上传到这 但是会解释我的思路:

```
D:\2024nian\java\lab2>java MaxOf 90 87 6 54 3
54

D:\2024nian\java\lab2>_
```

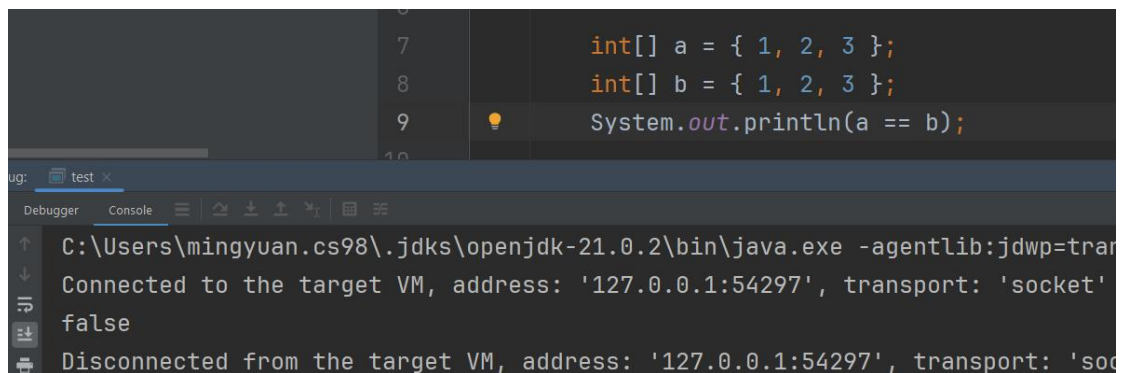
1. 我的代码首先把命令行的字符串转换 `int` 整形然后存储到一个新数组;
2. 然后通过 `findMax` 自定义的一个方法计算最大值;
3. 在 `findMax` 方法另有一个 `max` 方法这个方法的作用是每次从给定小标开始找到最大值;
4. 找到最大值把它跟前面的下表换位置;
每次找到最大值放到前面然后从下一个小标开始找最大值;
这个是选择排序的思想;
然后在每次找到最大值时有一个 `cnt` 变量会++ 这个变量等于 3 时是我们返回 `arr[2]` 这样可以找到第三个最大值;

8

8. 以下程序的运行结果是什么？

```
int[] a = { 1, 2, 3 };  
int[] b = { 1, 2, 3 };  
System.out.println(a == b);
```

他们当然是不等于的虽然他们的 `value` 是一样的 因为 `arr` 是包装类的每次创建哪怕他们的值一样但是他们会指向不同的 `memory` 地址：



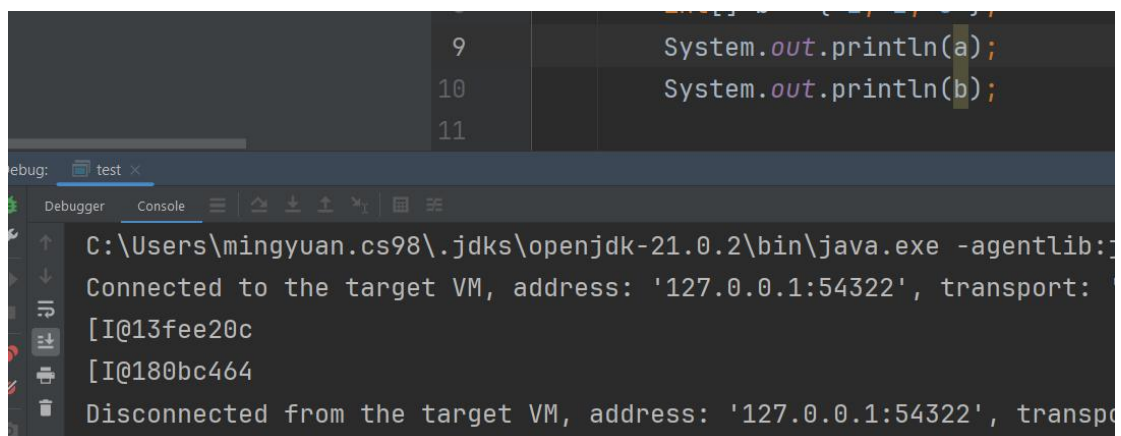
The screenshot shows a code editor with the following Java code:

```
int[] a = { 1, 2, 3 };  
int[] b = { 1, 2, 3 };  
System.out.println(a == b);
```

Below the code editor is a console window showing the execution output:

```
C:\Users\mingyuan.cs98\jdk\openjdk-21.0.2\bin\java.exe -agentlib:jdwp=tran  
Connected to the target VM, address: '127.0.0.1:54297', transport: 'socket'  
false  
Disconnected from the target VM, address: '127.0.0.1:54297', transport: 'soc
```

我们现在验证上面说的话：



The screenshot shows a code editor with the following Java code:

```
int[] a = { 1, 2, 3 };  
int[] b = { 1, 2, 3 };  
System.out.println(a);  
System.out.println(b);
```

Below the code editor is a console window showing the execution output:

```
C:\Users\mingyuan.cs98\jdk\openjdk-21.0.2\bin\java.exe -agentlib:jdwp=tran  
Connected to the target VM, address: '127.0.0.1:54322', transport: 'socket'  
[I@13fee20c  
[I@180bc464  
Disconnected from the target VM, address: '127.0.0.1:54322', transport: 'soc
```

我们会发现他们指向不一样的地址所以 `==` 会比较他们的地址这样会输出不一样的；
现在我们再次验证指向同一个地址的是等价如下代码：

```
6
7      int[] a = { 1, 2, 3 };
8      int []c = a;
9      System.out.println(a==c);
10

bug: test x
Debugger Console
C:\Users\mingyuan.cs98\jdk\openjdk-21.0.2\bin\java.exe -agentlib:jdwp=trans
Connected to the target VM, address: '127.0.0.1:54332', transport: 'socket'
true
Disconnected from the target VM, address: '127.0.0.1:54332', transport: 'sock
```

我们把 c 也指向 a 这样就会输出他们是等价的；

13. 8 皇后问题. 我们可以用排列代表国际象棋棋盘上皇后的位置. 例如排列 (52431) 可以代表如下棋盘的布局

```
***Q*
*Q***
****Q
**Q**
Q****
```

解法:

13.1 用一个数组来存储用户输入的皇后位置, 如上面例子数组会存储 **5 2 4 3 1**

```
Scanner scan = new Scanner(System.in);
int[] arr = new int[100]; // 存放Q的位置
int i = 0;
int cnt = scan.nextInt();

// 用户输入Q的位置
while (cnt != 0) {
    arr[i++] = cnt;
    cnt = scan.nextInt();
}
```

13.2 我们用一个 2 为字符串数组来把每个皇后存放到自己位置。

皇后位置是数组每个元素代表行，而 j 代表相应的列

比如 5 代表皇后位置在 5 行 1 列，因为 5 是第一个元素所以 1 列，其他元素以此类推都存放到自己的位置

```
// 把Q存放到自己的位置
char[][] table = new char[i][i];
for (int j = 0; j < i; j++) {
    int curr = arr[j];
    table[curr - 1][j] = 'Q';
}
```

13.3 写一个函数来判断皇后位置是否安全：

我分成 4 个情况来判断皇后位置是否准确，第一种是每行不能有两个皇后：

```
// 判断皇后位置是否安全
public static boolean isSafe(char[][] table) {
    int n = table.length;

    // 检查每一行是否只有一个皇后
    for (int i = 0; i < n; i++) {
        int count = 0;
        for (int j = 0; j < n; j++) {
            if (table[i][j] == 'Q') {
                count++;
                if (count > 1) {
                    return false; // 不安全
                }
            }
        }
    }
}
```

13.4 检查每一列是否只有一个皇后:

```
// 检查每一列是否只有一个皇后
for (int j = 0; j < n; j++) {
    int count = 0;
    for (int i = 0; i < n; i++) {
        if (table[i][j] == 'Q') {
            count++;
            if (count > 1) {
                return false; // 不安全
            }
        }
    }
}
```

13.5 检查主对角线:

```
// 检查主对角线是否只有一个皇后
for (int k = 0; k < n * 2 - 1; k++) {
    int count = 0;
    for (int i = 0; i < n; i++) {
        int j = k - i;
        if (j >= 0 && j < n) {
            if (table[i][j] == 'Q') {
                count++;
                if (count > 1) {
                    return false; // 不安全
                }
            }
        }
    }
}
```

13.6 检查次对角线:

```

// 检查次对角线是否只有一个皇后
for (int k = 1 - n; k < n; k++) {
    int count = 0;
    for (int i = 0; i < n; i++) {
        int j = i + k;
        if (j >= 0 && j < n) {
            if (table[i][j] == 'Q') {
                count++;
                if (count > 1) {
                    return false; // 不安全
                }
            }
        }
    }
}
}

```

每一种情况分你检查如果一种情况是假的会返回假，否子返回真

```

boolean safe = isSafe(table);

if (safe)
    System.out.println("Safe");
else
    System.out.println("Not Safe");

```


10. 给定整数 N ，输出 1 到 N 的所有排列。利用本题测试第 7 题中寻找中位数的算法是否正确。

假设我们的数组只有 1...3 的数字，通过下面代码我们生命，且初始话数组：

```
public static void main(String[] args) {  
    int N = 3; // 假设N = 1...10  
    int[] nums = new int[N];  
    for (int i = 0; i < N; i++) {  
        nums[i] = i + 1; //初始化  
    }  
}
```

10.1 通过递归来实现 1 到 N 的排列

```
// 生成1到N的所有排列  
public static void generatePermutations(int[] nums, int index) {  
    if (index == nums.length) { //递归的终止条件  
        printArray(nums);  
        return;  
    }  
  
    for (int i = index; i < nums.length; i++) {  
        swap(nums, index, i); //交换  
        generatePermutations(nums, index + 1); //递归  
        swap(nums, index, i);  
    }  
}
```

输出结果：

```
1 2 3  
1 3 2  
2 1 3  
2 3 1  
3 2 1  
3 1 2  
Disconnected f
```

14. 给定 N, 输出 N 阶蛇型矩阵 分别是 s (1) .s (3)

s (1) = [1] s(3) = 1 2 3
 8 9 4
 7 6 5

分析首先创建一个 N×N 的矩阵，并用 0 初始化。然后按照贪蛇的填充规则，从左上角开始填充数字，依次向右、向下、向左、向上填充，直到矩阵填满。最后打印出填充完成的贪蛇矩阵。

初始化：

```
int MAX_SIZE = 100;
int n, i, j, num = 1;
int [][]matrix = new int[MAX_SIZE][MAX_SIZE];

// 用户输入N
System.out.println("Enter the size of the snake matrix "+ MAX_SIZE);
Scanner scan = new Scanner(System.in);
n = scan.nextInt();

// 初始化矩阵
for (i = 0; i < n; i++) {
    for (j = 0; j < n; j++) {
        matrix[i][j] = 0;
    }
}
```

```
// 填充贪蛇
int row = 0, col = 0;
while (num <= n * n) {
    // 向右填充
    while (col < n && matrix[row][col] == 0) {
        matrix[row][col++] = num++;
    }
    col--; // 回退到上一个位置
    row++; // 向下移动一行

    // 向下填充
    while (row < n && matrix[row][col] == 0) {
        matrix[row++][col] = num++;
    }
    row--; // 回退到上一个位置
    col--; // 向左移动一列
```

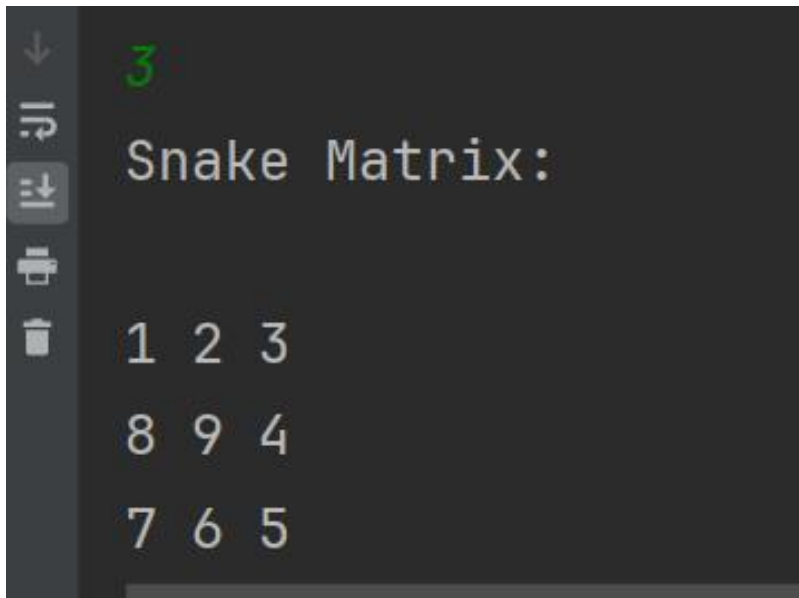
```

// 向左填充
while (col >= 0 && matrix[row][col] == 0) {
    matrix[row][col--] = num++;
}
col++; // 回退到上一个位置
row--; // 向上移动一行

// 向上填充
while (row >= 0 && matrix[row][col] == 0) {
    matrix[row--][col] = num++;
}
row++; // 回退到上一个位置
col++; // 向右移动一列

```

输出结果:



```

3
Snake Matrix:
1 2 3
8 9 4
7 6 5

```

15. 编写库文件 **Statistic.java**, 包含如下静态方法:

15.1 **max** 返回数组 **a** 中最大值;

```

public static double max(double a[]){
    double max = a[0];
    for(int i=0;i<a.length;i++){
        if(a[i]>max){
            max = a[i];
        }
    }
    return max;
}

```

15.2 min 返回数组 a 中最小值：

```
public static double min(double a[]){
    double min = a[0];
    for(int i=0;i<a.length;i++){
        if(a[i]<min)
            min = a[i];
    }
    return min;
}
```

15.3 返回 a 数组均值：

```
public static double mean(double a[]){
    double sum = 0;
    for(int i=0;i<a.length;i++){
        sum+=a[i];
    }
    return sum / a.length;
}
```

15.4 返回 a 数组方差公式如下：

$$s^2 = \frac{\sum (x_i - \bar{x})^2}{n - 1}$$

```
public static double variance(double a[]){
    double sum = 0;
    for(int i=0;i<a.length;i++){
        sum=sum+Math.pow(a[i]-mean(a),2);
    }
    return (sum / (a.length-1));
}
```

15.5 返回 a 数组中第 k 大解释在第 7 道题解说思路（即选择排序的思路）：

```
public static double findKMax(double[] a,int k) {  
    int max_index = 0,cnt = 0;  
    for(int i= 0;i<a.length;i++){  
        max_index = max(i,a);  
        cnt++;  
        if(cnt==k)  
            return a[max_index];  
  
        double tmp = a[i];  
        a[i] = a[max_index];  
        a[max_index] = tmp;  
    }  
    return -1;  
}
```

```
public static int max(int index, double[] arr) {  
    int maxOf = index;  
    for(int i = index;i<arr.length;i++){  
        if(arr[i]>arr[maxOf])  
            maxOf = i;  
    }  
    return maxOf;  
}
```