* 处理输入测试用例：

import java.util.\*;

public Main{

public static void main(String[] args){

Scanner scanner = new Scanner(System.in);

while(scanner.hasNext()){

//代码

}

}

1. 获取下一行字符串

String str = scanner.nextLine();

1. 获取下一个整数

Int I = scanner.nextInt();、

1. 获取下一个long

**Long l = Long.parseLong(scanner.next())**

* Array、List、Set互相转化
  1. Array转List：Arrays.asList()
  2. List转Array：list.toArray(new 类型[大小])
  3. List和Set间相互转化：构造函数
  4. Array转Set：先转换成list再转换成Set
  5. Set转Array：同list转Array
* 排序：

1. 列表：
   1. 升序：Collections.sort(list)
   2. 降序：先升序再反转：Collections.reverse(list)
2. 数组
   1. 升序：Arrays.sort(arr)
   2. 降序：通过Arrays.asList()转换为列表，逆序后再转换为数组

* Collections工具函数
  + reverse(list)
  + sort(list)
  + swap(list，index1， index2)
* 数组函数
  1. 长度：.length
  2. 拷贝：System.arraycopy(原数组，原数组开始位置，目标数组，目标数组开始位置，长度)
  3. 转换为字符串：Arrays.toString(arr)//格式为[元素，元素，。。。]
  4. 二分查找：Arrays.binarySearch(arr，（fromIndexd， toIndex），target)
     + 若存在，返回索引（非负数）
     + 若不存在，但在范围内，由1开始计数，得“ - 插入点索引值”
     + 若不存在，且小于所有元素，返回-1或-（fromIndex-1），否则返回– (length + 1)或-（toIndex + 1）
  5. 全部填充：Arrays.fill(array，val)
* 字符函数：Character.
  1. 判断类型
     + isDigit
     + isLetter
     + isLetterOrDigit
     + isLowerCase
     + isUpperCase
     + isWhitespace
  2. 获取Unicode代码：codePointAt(chararr/string, index)，<256为ASCII字符，大小为1字节，>255为2字节
  3. 转换为字符串:toString()
* 字符串函数
  1. 由字符数组构造：String(chararr(, beg, end，指定开始和结尾索引))
  2. 字串：substring（开始索引，结束索引——若无则默认为到结尾）——返回新对象
  3. 是否包含子字符串：contains
  4. 访问字符：charAt(index)
  5. 转大小写：toUpperCase(),toLowerCase()——返回新对象
  6. 匹配正则表达式：matches(regex)
  7. 分割、替换字符串：split(regex)，replace(regex)
     + 用“.”分割、替换：”[\\.](file:///\\.)”，
     + 用”\*”分割、替换：”\\\*”
     + 用多个空格分隔：“\\s+“
     + 用“\”分割：“\\\\”
  8. 和字符数组相互转化：
     + 字符串->数组：string.toCharArray()
     + 数组->字符串：new String(array)
  9. StringBuilder类
     + 构造：new StringBuilder(空，或String)
     + 添加字符：append(char)
     + 翻转：reverse()——**用于反转字符串！**
     + 转换为字符串：toString
     + indexOf，lastIndexOf
     + delete，deleteCharAt
     + setCharAt
     + substring
     + replace
  10. 是否以某个字符串开头：startsWith(String)
  11. 是否以某个字符串结束：endsWith(String)
  12. 获取子字符或字符串
      + 首次出现索引：indexOf(char/String，开始位置)，不存在返回-1
      + 最后一次出现索引：lastIndexOf
  13. 去除前后空格：trim()
  14. 替换：replace(旧字符，新字符)，replace(旧子串，新子串)，replaceAll————**返回新对象！**
  15. 转换为整数：Integer.parseInt(str，字符串代表的数字的进制，默认为10)————**若报错可能是因为该数值太大超过int，需用Long.parseLong()代替！**
  16. 比较大小：compareTo(other)，用字典序比较，大于则>0，等于则=0，小于则<0
* 列表函数
  1. 访问元素：**get**(index)
  2. 删除元素：remove(index)——**遍历中应该用迭代器删除！**
  3. 判断包含：cotains(value)
  4. 添加元素：add(**index**，若无则默认为最后，value)
  5. **获取最值**
     + Collections.max
     + Collections.min
  6. LinkedList实现栈和队列
     + 栈
       - 入栈：list.**addFirst**()
       - 出栈：list.**removeFirst**()
       - 栈顶：list.getFirst()
     + 队列
       - 入队：list.**addLast**()
       - 出队：list.**removeFirst**()
       - 队首：list.getFirst()
* 哈希表
  1. 若key用放入顺序存储则选用LinedHashMap
  2. 若按key排序放入，则选用TreeMap
* 栈：Stack<> stack = new Stack<>()
  1. push
  2. peek
  3. pop
* 队列：Queue<> queue = new **LinkedList**<>()
  1. offer
  2. peek
  3. poll
* 集合
  1. 类型
     + TreeSet
       - add
       - pollFirst
       - pollLast
     + HashSet
     + LinkedHashSet
  2. 操作
     + add
     + iterator()：获取迭代器
* 优先队列PriorityQueue
  1. 构造：指定大小和比较规则

Queue<Integer> pq = new PriorityQueue<Integer>(11,new Comparator<Integer>() {

public int compare(Integer i1, Integer i2) {

return i2 - i1;//降序，默认为升序

}

});

* 1. 添加：add
  2. 队首：peek
  3. 出队：poll
  4. 大小：size
  5. 遍历：转换为数组：toArray
* 类型转换:double转int：（int）double
* 遍历
  1. 索引
  2. 迭代：”:”运算符，适用于数组，列表，集合
* 获取字符对应ASCII编码：Integer.valueOf(char c);
* 注意：在main函数中使用内部自定义类、成员和方法，必须将自定义类声明为static
* 常见转义字符：\\., \\, \’, \”, \r, \n
* 整数
  1. 字面值默认为int类型，指定为long需加l或L后缀
  2. 字符串转整数：Integer/Long.parseInt(字符串，字符串对应进制)
  3. 进制转换
     + 10进制转2进制 Integer/Long.toBinaryString(n); 一个二进制字符串.
     + 10进制转8进制 Integer/Long.toOctalString(n); 一个八进制字符串
     + 10进制转16进制 Integer/Long.toHexString(n); 一个16进制字符串
     + 10进制转 r 进制 Integer/Long.toString(100, r); 一个r进制字符串
  4. 最值：
     + Integer/Long.MAX\_VALUE
     + Integer/Long.MIN\_VALUE
* 浮点数
  1. 浮点数字面值默认为double类型，指定为float需加f或F后缀
  2. 精度控制
     + import java.text.DecimalFormat;
     + System.out.println(new DecimalFormat("0.00").format(pi))
  3. 高精度浮点数：BigDecimal
* 正则表达式：要用特殊字符需要转义：\\特殊字符
  1. ‘.’：匹配任意字符，要用字符本身则转义：\\.
  2. ‘^’：匹配一行的开头，在[]中表示对后面字符的反向匹配
  3. ‘$‘：匹配一行的结尾
  4. {m，n}：定义一个出现次数范围
  5. []：定义一个可选字符集合，用-连接字符则表示范围
  6. ()：定义一个捕获组
  7. ‘\*’：出现0或任意次
  8. ‘+’：出现zhi少一次
  9. ‘?’：出现0或1次，紧跟在其他限定符后表示非贪心匹配，即匹配最短的字符串，贪心模式匹配最长字符串
  10. (?:*pattern*)：定义一个非捕获匹配组
  11. (?=*pattern*)：执行正向预测先行搜索的子表达式，是一个非捕获匹配。例如，'Windows (?=95|98|NT|2000)' 匹配"Windows 2000"中的"Windows"，但不匹配"Windows 3.1"中的"Windows"
  12. (?!*pattern*)：执行反向预测先行搜索的子表达式，例如，'Windows (?!95|98|NT|2000)' 匹配"Windows 3.1"中的 "Windows"，但不匹配"Windows 2000"中的"Windows"
  13. ‘pattern1|pattern2‘：匹配pattern1或pattern2
  14. *pattern*\b：匹配一个字边界，即字与空格间的位置。例如，"er\b"匹配"never"中的"er"，但不匹配"verb"中的"er"
  15. *pattern*\B: 非字边界匹配。"er\B"匹配"verb"中的"er"，但不匹配"never"中的"er"
  16. \d：匹配数字字符
  17. \D：匹配非数字字符
  18. \n：匹配换行符
  19. \r：匹配回车符
  20. \s：匹配任何空白符，包括空格、制表符、换页符等，与 [ \f\n\r\t\v] 等效
  21. \S：匹配任何非空白字符。与 [^ \f\n\r\t\v] 等效。
  22. \w：匹配任何字类字符，包括下划线。与"[A-Za-z0-9\_]"等效。
  23. \W：与任何非单词字符匹配。与"[^A-Za-z0-9\_]"等效。