目录

JAVA 操作

String 操作

字符串查找

String 提供了两种查找字符串的方法，即 indexOf 与 lastIndexOf 方法。

1、indexOf（String s）

该方法用于返回参数字符串 s 在指定字符串中首次出现的索引位置，当调用字符串的 indexOf()方法时，会从当前字符串的开始位置搜索 s 的位置；如果没有检索到字符串 s，该方法返回-1

2、lastIndexOf(String str)

该方法用于返回字符串最后一次出现的索引位置。当调用字符串的 lastIndexOf()方法时，会从当前字符串的开始位置检索参数字符串 str，并将最后一次出现 str 的索引位置返回。如果没有检索到字符串 str，该方法返回-1.

如果 lastIndexOf 方法中的参数是空字符串"" ，，则返回的结果与 length 方法的返回结果相同。

获取指定索引位置的字符

使用 charAt()方法可将指定索引处的字符返回。

```
1 String str = "hello word";
2 char mychar =   str.charAt(5);   // mychar 的结果是 w
```

获取子字符串

通过 String 类的 substring()方法可对字符串进行截取。这些方法的共同点就是都利用字符串的下标进行截取，且应明确字符串下标是从 0 开始的。在字符串中空格占用一个索引位置。

1、substring(int beginIndex)

该方法返回的是从指定的索引位置开始截取知道该字符串结尾的子串。

```
1 String str = "Hello word";
2 String substr = str.substring(3); //获取字符串，此时 substr 值为 lo word
```

2、substring(int beginIndex, int endIndex)

beginIndex：开始截取子字符串的索引位置

endIndex：子字符串在整个字符串中的结束位置

```
1 String str = "Hello word";
2 String substr = str.substring(0,3); //substr 的值为 hel
```

去除空格

trim()方法返回字符串的副本，忽略前导空格和尾部空格。

字符串替换

replace()方法可实现将指定的字符或字符串替换成新的字符或字符串

oldChar：要替换的字符或字符串

newChar：用于替换原来字符串的内容

如果要替换的字符 oldChar 在字符串中重复出现多次，replace()方法会将所有 oldChar 全部替换成 newChar。需要注意的是，要替换的字符 oldChar 的大小写要与原字符串中字符的大小写保持一致。

判断字符串的开始与结尾

startsWith()方法与 endsWith()方法分别用于判断字符串是否以指定的内容开始或结束。这两个方法的返回值都为 boolean 类型。

1、startsWith(String prefix)

该方法用于判断当前字符串对象的前缀是否是参数指定的字符串。

2、endsWith(String suffix)

该方法用于判断当前字符串是否以给定的子字符串结束

判断字符串是否相等

1、equals(String otherstr)

如果两个字符串具有相同的字符和长度，则使用 equals()方法比较时，返回 true。同时 equals()方法比较时区分大小写。

2、equalsIgnoreCase(String otherstr)

equalsIgnoreCase()方法与 equals()类型，不过在比较时忽略了大小写。

按字典顺序比较两个字符串

compareTo()方法为按字典顺序比较两个字符串，该比较基于字符串中各个字符的 Unicode 值，按字典顺序将此 String 对象表示的字符序列与参数字符串所表示的字符序列进行比较。如果按字典顺序此 String 对象位于参数字符串之前，则比较结果为一个负整数；如果按字典顺序此 String 对象位于参数字符串之后，则比较结果为一个正整数；如果这两个字符串相等，则结果为 0.

```
1 str.compareTo(String otherstr);
```

字母大小写转换

字符串的 toLowerCase()方法可将字符串中的所有字符从大写字母改写为小写字母，而 tuUpperCase()方法可将字符串中的小写字母改写为大写字母。

```
1 str.toLowerCase();
2 str.toUpperCase();
```

字符串分割

使用 split()方法可以使字符串按指定的分隔字符或字符串对内容进行分割，并将分割后的结果存放在字符数组中。
1 str.split(String sign);
sign 为分割字符串的分割符，也可以使用正则表达式。
没有统一的对字符串进行分割的符号，如果想定义多个分割符，可使用符号“|”。例如，“,|=”表示分割符分别为“，”和“=”。
1 str.split(String sign, in limit)；
该方法可根据给定的分割符对字符串进行拆分，并限定拆分的次数。

字符串操作手册

char charAt(int index)
返回指定索引处的 char 值。

int compareTo(Object o)
把这个字符串和另一个对象比较。

int compareTo(String anotherString)
按字典顺序比较两个字符串。

int compareToIgnoreCase(String str)
按字典顺序比较两个字符串，不考虑大小写。

String concat(String str)
将指定字符串连接到此字符串的结尾。

boolean contentEquals(StringBuffer sb)
当且仅当字符串与指定的 StringButter 有相同顺序的字符时候返回真。

static String copyValueOf(char[] data)
返回指定数组中表示该字符序列的 String。

static String copyValueOf(char[] data, int offset, int count)
返回指定数组中表示该字符序列的 String。

boolean endsWith(String suffix)
测试此字符串是否以指定的后缀结束。

boolean equals(Object anObject)
将此字符串与指定的对象比较。

boolean equalsIgnoreCase(String anotherString)
将此 String 与另一个 String 比较，不考虑大小写。

byte[] getBytes()
使用平台的默认字符集将此 String 编码为 byte 序列，并将结果存储到一个新的 byte 数组中。

byte[] getBytes(String charsetName)
使用指定的字符集将此 String 编码为 byte 序列，并将结果存储到一个新的 byte 数组中。

void getChars(int srcBegin, int srcEnd, char[] dst, int dstBegin)
将字符从此字符串复制到目标字符数组。

int hashCode()
返回此字符串的哈希码。

int indexOf(int ch)
返回指定字符在此字符串中第一次出现处的索引。

int indexOf(int ch, int fromIndex)
返回在此字符串中第一次出现指定字符处的索引，从指定的索引开始搜索。

int indexOf(String str)
返回指定子字符串在此字符串中第一次出现处的索引。

int indexOf(String str, int fromIndex)
返回指定子字符串在此字符串中第一次出现处的索引，从指定的索引开始。

String intern()
返回字符串对象的规范化表示形式。

int lastIndexOf(int ch)
返回指定字符在此字符串中最后一次出现处的索引。

int lastIndexOf(int ch, int fromIndex)
返回指定字符在此字符串中最后一次出现处的索引，从指定的索引处开始进行反向搜索。

int lastIndexOf(String str)
返回指定子字符串在此字符串中最右边出现处的索引。

int lastIndexOf(String str, int fromIndex)
返回指定子字符串在此字符串中最后一次出现处的索引，从指定的索引开始反向搜索。

int length()
返回此字符串的长度。

boolean matches(String regex)
告知此字符串是否匹配给定的正则表达式。

boolean regionMatches(boolean ignoreCase, int toffset, String other, int ooffset, int len)
测试两个字符串区域是否相等。

boolean regionMatches(int toffset, String other, int ooffset, int len)
测试两个字符串区域是否相等。

String replace(char oldChar, char newChar)
返回一个新的字符串，它是通过用 newChar 替换此字符串中出现的所有 oldChar 得到的。

String replaceAll(String regex, String replacement
使用给定的 replacement 替换此字符串所有匹配给定的正则表达式的子字符串。

String replaceFirst(String regex, String replacement)
使用给定的 replacement 替换此字符串匹配给定的正则表达式的第一个子字符串。

String[] split(String regex)
根据给定正则表达式的匹配拆分此字符串。

String[] split(String regex, int limit)
根据匹配给定的正则表达式来拆分此字符串。

boolean startsWith(String prefix)
测试此字符串是否以指定的前缀开始。

boolean startsWith(String prefix, int toffset)
测试此字符串从指定索引开始的子字符串是否以指定前缀开始。

CharSequence subSequence(int beginIndex, int endIndex)
返回一个新的字符序列，它是此序列的一个子序列。

String substring(int beginIndex)
返回一个新的字符串，它是此字符串的一个子字符串。

String substring(int beginIndex, int endIndex)
返回一个新字符串，它是此字符串的一个子字符串。

char[] toCharArray()
将此字符串转换为一个新的字符数组。

String toLowerCase()
使用默认语言环境的规则将此 String 中的所有字符都转换为小写。

String toLowerCase(Locale locale)
使用给定 Locale 的规则将此 String 中的所有字符都转换为小写。

String toString()
返回此对象本身（它已经是一个字符串 !)。

String toUpperCase()
使用默认语言环境的规则将此 String 中的所有字符都转换为大写。

String toUpperCase(Locale locale)
使用给定 Locale 的规则将此 String 中的所有字符都转换为大写。

String trim()
返回字符串的副本，忽略前导空白和尾部空白。

static String valueOf(primitive data type x)
返回给定 data type 类型 x 参数的字符串表示形式。

输入输出

输入函数

boolean	**hasNextBigDecimal**() 如果通过使用 nextBigDecimal() 方法，此扫描器输入信息中的下一个标记可以解释为默认基数中的一个 BigDecimal，则返回 true。
boolean	**hasNextBigInteger**() 如果通过使用 nextBigInteger() 方法，此扫描器输入信息中的下一个标记可以解释为默认基数中的一个 BigInteger 值，则返回 true。
boolean	**hasNextBigInteger**(int radix) 如果通过使用 nextBigInteger() 方法，此扫描器输入信息中的下一个标记可以解释为指定基数中的一个 BigInteger 值，则返回 true。
boolean	**hasNextBoolean**() 如果通过使用一个从字符串 "true|false" 创建的大小写敏感的模式，此扫描器输入信息中的下一个标记可以解释为一个布尔值，则返回 true。
boolean	**hasNextByte**() 如果通过使用 nextByte() 方法，此扫描器输入信息中的下一个标记可以解释为默认基数中的一个字节值，则返回 true。
boolean	**hasNextByte**(int radix) 如果通过使用 nextByte() 方法，此扫描器输入信息中的下一个标记可以解释为指定基数中的一个字节值，则返回 true。
boolean	**hasNextDouble**() 如果通过使用 nextDouble() 方法，此扫描器输入信息中的下一个标记可以解释为默认基数中的一个 double 值，则返回 true。
boolean	**hasNextFloat**() 如果通过使用 nextFloat() 方法，此扫描器输入信息中的下一个标记可以解释为默认基数中的一个 float 值，则返回 true。
boolean	**hasNextInt**() 如果通过使用 nextInt() 方法，此扫描器输入信息中的下一个标记可以解释为默认基数中的一个 int 值，则返回 true。
boolean	**hasNextInt**(int radix) 如果通过使用 nextInt() 方法，此扫描器输入信息中的下一个标记可以解释为指定基数中的一个 int 值，则返回 true。
boolean	**hasNextLine**() 如果在此扫描器的输入中存在另一行，则返回 true。
boolean	**hasNextLong**() 如果通过使用 nextLong() 方法，此扫描器输入信息中的下一个标记可以解释为默认基数中的一个 long 值，则返回 true。
boolean	**hasNextLong**(int radix) 如果通过使用 nextLong() 方法，此扫描器输入信息中的下一个标记可以解释为指定基数中的一个 long 值，则返回 true。
boolean	**hasNextShort**() 如果通过使用 nextShort() 方法，此扫描器输入信息中的下一个标记可以解释为默认基数中的一个 short 值，则返回 true。
boolean	**hasNextShort**(int radix) 如果通过使用 nextShort() 方法，此扫描器输入信息中的下一个标记可以解释为指定基数中的一个 short 值，则返回 true。

String	**next**() 查找并返回来自此扫描器的下一个完整标记。
String	**next**(Pattern pattern) 如果下一个标记与指定模式匹配，则返回下一个标记。
String	**next**(String pattern) 如果下一个标记与从指定字符串构造的模式匹配，则返回下一个标记。
BigDecimal	**nextBigDecimal**() 将输入信息的下一个标记扫描为一个 BigDecimal。
BigInteger	**nextBigInteger**() 将输入信息的下一个标记扫描为一个 BigInteger。
BigInteger	**nextBigInteger**(int radix) 将输入信息的下一个标记扫描为一个 BigInteger。
boolean	**nextBoolean**() 扫描解释为一个布尔值的输入标记并返回该值。
byte	**nextByte**() 将输入信息的下一个标记扫描为一个 byte。
byte	**nextByte**(int radix) 将输入信息的下一个标记扫描为一个 byte。
double	**nextDouble**() 将输入信息的下一个标记扫描为一个 double。
float	**nextFloat**() 将输入信息的下一个标记扫描为一个 float。
int	**nextInt**() 将输入信息的下一个标记扫描为一个 int。
int	**nextInt**(int radix) 将输入信息的下一个标记扫描为一个 int。
String	**nextLine**() 此扫描器执行当前行，并返回跳过的输入信息。
long	**nextLong**() 将输入信息的下一个标记扫描为一个 long。
long	**nextLong**(int radix) 将输入信息的下一个标记扫描为一个 long。
short	**nextShort**() 将输入信息的下一个标记扫描为一个 short。
short	**nextShort**(int radix) 将输入信息的下一个标记扫描为一个 short。

printf, println, print

```
public class TestPrintf {
public static void main(String[] args) {
//定义一些变量，用来格式化输出。
double d = 345.678;
String s = "你好！";
int i = 1234;
//"%"表示进行格式化输出，"%"之后的内容为格式的定义。
System.out.printf("%f",d); //"f"表示格式化输出浮点数。
System.out.println();
System.out.printf("%9.2f",d); //"9.2"中的 9 表示输出的长度，2 表示小数点后的位数。
System.out.println();
System.out.printf("%+9.2f",d); //"+"表示输出的数带正负号。
System.out.println();
System.out.printf("%-9.4f",d); //"-"表示输出的数左对齐（默认为右对齐）。
System.out.println();
System.out.printf("%+-9.3f",d); //"+-"表示输出的数带正负号且左对齐。
System.out.println();
System.out.printf("%d",i); //"d"表示输出十进制整数。
System.out.println();
System.out.printf("%o",i); //"o"表示输出八进制整数。
System.out.println();
System.out.printf("%x",i); //"d"表示输出十六进制整数。
System.out.println();
System.out.printf("%#x",i); //"d"表示输出带有十六进制标志的整数。
System.out.println();
System.out.printf("%s",s); //"d"表示输出字符串。
System.out.println();
System.out.printf("输出一个浮点数：%f，一个整数：%d，一个字符串：%s",d,i,s);
//可以输出多个变量，注意顺序。
System.out.println();
System.out.printf("字符串：%2$s，%1$d 的十六进制数：%1$#x",i,s);
//"X$"表示第几个变量。
}
}
```

例子

```
 1 mport java.util.Scanner;
 2 public class Example
 3 {
 4     public static void main(String args[])
 5     {
 6         System.out.println("请输入若干个数,每输入一个数用回车确认");
 7         System.out.println("最后输入一个非数字结束输入操作");
 8         Scanner reader=new Scanner(System.in);
 9         double sum=0;
10         int m=0;
11         while(reader.hasNextDouble())
12         {
13             double x=reader.nextDouble();
14             m=m+1;
15             sum=sum+x;
16         }
17         System.out.printf("%d个数的和为%f\n",m,sum);
18         System.out.printf("%d个数的平均值是%f\n",m,sum/m);
19     }
20 }
```

BigInteger

Ⅰ基本函数：

1.valueOf(parament); 将参数转换为制定的类型
比如 int a=3;
BigInteger b=BigInteger.valueOf(a);
则 b=3;
String s="2345"
BigInteger c=BigInteger.valueOf(s);
则 c=12345；

2.add(); 大整数相加

```
BigInteger a=new BigInteger("3");
BigInteger b=new BigInteger("4");
c = a.add(b);
```

3.subtract(); 相减
4.multiply(); 相乘
5.divide(); 相除取整
6.remainder(); 取余
7.pow(); a.pow(b)=a^b
8.gcd(); 最大公约数

9.abs(); 绝对值
10.negate(); 取反数
11.mod(); a.mod(b)=a%b=a.remainder(b);
12.max(); min();
13.punlic int comareTo();
14.boolean equals(); 是否相等
15.BigInteger 构造函数：
一般用到以下两种：
BigInteger(String val);
将指定字符串转换为十进制表示形式；
BigInteger(String val,int radix);
将指定基数的 BigInteger 的字符串表示形式转换为 BigInteger

II.基本常量：

A=BigInteger.ONE 1
B=BigInteger.TEN 10
C=BigInteger.ZERO 0
III.基本操作
1. 读入：
用 Scanner 类定义对象进行控制台读入,Scanner 类在 java.util.*包中
Scanner cin=new Scanner(System.in);// 读入
while(cin.hasNext()) //等同于!=EOF
{
int n;
BigInteger m;
n=cin.nextInt(); //读入一个 int;
m=cin.BigInteger();//读入一个 BigInteger;
System.out.print(m.toString());
}

IV.运用

四则预算：

```
import java.util.Scanner;
import java.math.*;
import java.text.*;
public class Main
{
    public static void main(String args[])
    {
        Scanner cin = new Scanner ( System.in );
```

```
BigInteger a,b;
int c;
char op;
String s;

while( cin.hasNext() )
{
	a = cin.nextBigInteger();
	s = cin.next();
	op = s.charAt(0);
	if( op == '+')
	{
		b = cin.nextBigInteger();
		System.out.println(a.add(b));
	}
	else if( op == '-')
	{
		b = cin.nextBigInteger();
		System.out.println(a.subtract(b));
	}
	else if( op == '*')
	{
		b = cin.nextBigInteger();
		System.out.println(a.multiply(b));
	}
	else
	{
		BigDecimal a1,b1,eps;
		String s1,s2,temp;
		s1 = a.toString();
		a1 = new BigDecimal(s1);
		b = cin.nextBigInteger();
		s2 = b.toString();
		b1 = new BigDecimal(s2);
		c = cin.nextInt();
		eps = a1.divide(b1,c,4);
		//System.out.println(a + " " + b + " " + c);
		//System.out.println(a1.doubleValue() + " " + b1.doubleValue() + " " + c);
		System.out.print( a.divide(b) + " " + a.mod(b) + " ");
		if( c != 0)
		{
			temp = "0.";
			for(int i = 0; i < c; i ++) temp += "0";
			DecimalFormat gd = new DecimalFormat(temp);
```

```
                    System.out.println(gd.format(eps));
                }
                else System.out.println(eps);
            }
        }
    }
}
```

补充：

```
a=a.pow(b);
a=a.stripTrailingZeros();
d=a.toPlainString();
if(d.charAt(0)=='0') d=d.substring(1);
```

BigDecimal 例子

```
import  java.math.BigDecimal;
/**
 *  由于 Java 的简单类型不能够精确的对浮点数进行运算，这个工具类提供精
 *  确的浮点数运算，包括加减乘除和四舍五入。
 */
public  class  Arith{
    //默认除法运算精度
    private  static  final  int  DEF_DIV_SCALE  =  10;
    //这个类不能实例化
    private  Arith(){
    }
    /**
     *  提供精确的加法运算。
     *  @param  v1  被加数
     *  @param  v2  加数
     *  @return  两个参数的和
     */
    public  static  double  add(double  v1,double  v2){
        BigDecimal  b1  =  new  BigDecimal(Double.toString(v1));
        BigDecimal  b2  =  new  BigDecimal(Double.toString(v2));
        return  b1.add(b2).doubleValue();
    }
    /**
     *  提供精确的减法运算。
     *  @param  v1  被减数
     *  @param  v2  减数
```

```
     *  @return  两个参数的差
     */
    public static double sub(double v1,double v2){
        BigDecimal b1 = new BigDecimal(Double.toString(v1));
        BigDecimal b2 = new BigDecimal(Double.toString(v2));
        return b1.subtract(b2).doubleValue();
    }
    /**
     *  提供精确的乘法运算。
     *  @param  v1  被乘数
     *  @param  v2  乘数
     *  @return  两个参数的积
     */
    public static double mul(double v1,double v2){
        BigDecimal b1 = new BigDecimal(Double.toString(v1));
        BigDecimal b2 = new BigDecimal(Double.toString(v2));
        return b1.multiply(b2).doubleValue();
    }
    /**
     *  提供（相对）精确的除法运算，当发生除不尽的情况时，精确到
     *  小数点以后 10 位，以后的数字四舍五入。
     *  @param  v1  被除数
     *  @param  v2  除数
     *  @return  两个参数的商
     */
    public static double div(double v1,double v2){
        return div(v1,v2,DEF_DIV_SCALE);
    }
    /**
     *  提供（相对）精确的除法运算。当发生除不尽的情况时，由 scale 参数指
     *  定精度，以后的数字四舍五入。
     *  @param  v1  被除数
     *  @param  v2  除数
     *  @param  scale  表示表示需要精确到小数点以后几位。
     *  @return  两个参数的商
     */

    public static double div(double v1,double v2,int scale){
        if(scale<0){
            throw new IllegalArgumentException(
                "The scale must be a positive integer or zero");
        }
        BigDecimal b1 = new BigDecimal(Double.toString(v1));
        BigDecimal b2 = new BigDecimal(Double.toString(v2));
```

```
            return  b1.divide(b2,scale,BigDecimal.ROUND_HALF_UP).doubleValue();
        }
        /**
         *  提供精确的小数位四舍五入处理。
         *  @param  v  需要四舍五入的数字
         *  @param  scale  小数点后保留几位
         *  @return  四舍五入后的结果
         */
        public  static  double  round(double  v,int  scale){
            if(scale<0){
                throw  new  IllegalArgumentException(
                    "The  scale  must  be  a  positive  integer  or  zero");
            }
            BigDecimal  b  =  new  BigDecimal(Double.toString(v));
            BigDecimal  one  =  new  BigDecimal("1");
            return  b.divide(one,scale,BigDecimal.ROUND_HALF_UP).doubleValue();
        }
    };
```

类型转换

string->byte
Byte static byte parseByte(String s)

byte->string
Byte static String toString(byte b)

char->string
Character static String to String (char c)

string->Short
Short static Short parseShort(String s)

Short->String
Short static String toString(Short s)

String->Integer
Integer static int parseInt(String s)

Integer->String
Integer static String tostring(int i)

String->Long
Long static long parseLong(String s)

Long->String
Long static String toString(Long i)

String->Float
Float static float parseFloat(String s)

Float->String
Float static String toString(float f)

String->Double
Double static double parseDouble(String s)

Double->String
Double static String toString(Double d)

编译运行
使用 javac 命令来编译 java 程序，格式为 javac ***.java（***为 java 文件名）
使用 java 命令来运行 java 程序，格式为 java ***，记住不要带扩展名

使用例程

```
import java.util.Scanner;
import java.math.*;
import java.text.*;

public class Main{
    public static void main(String args[]){
        Scanner cin = new Scanner(System.in);
        int T;
        T = cin.nextInt();
        BigInteger N = new BigInteger("0");
        BigInteger M = new BigInteger("0");
        for (int i = 0; i < T; ++i) {
            BigInteger BASE = new BigInteger("1");
            BigInteger ANS = new BigInteger("0");

            N = cin.nextBigInteger();
            M = cin.nextBigInteger();
            while (true) {
                BASE = BASE.multiply(new BigInteger("2"));
```

```
                if (BASE.compareTo(N) > 0) break;
            }

            while (true) {
                if (N.compareTo(BigInteger.ZERO) == 0) break;
                if (N.compareTo(BASE.subtract(new BigInteger("1")).multiply(M)) > 0) {
                    BigInteger K = N.divide(BASE);
                    K = K.min(M);
                    ANS = ANS.add(BASE);
                    N = N.subtract(K.multiply(BASE));
                }
                BASE = BASE.divide(new BigInteger("2"));
            }

            System.out.println(ANS);
        }
    }
}
```