# 求最小公倍数

题目描述

正整数A和正整数B 的最小公倍数是指 能被A和B整除的最小的正整数值，设计一个算法，求输入A和B的最小公倍数。

## 输入描述:

输入两个正整数A和B。

## 输出描述:

输出A和B的最小公倍数。

**示例1**

## 输入

5 7

## 输出

35

解析思路：

求最小公倍数，使用公式法 (a,b) \* [a,b] = a \* b

(a,b) 是求两个数的最大公约数

[a,b] 是求两个数的最小公倍数

a \* b 两个数的乘积等于最小公约数乘于最小公倍数

例如：

|  |
| --- |
| 5 \* 7 = (5,7) \* [5,7]  35 = 1 \* 35 |

求最大公约数，使用辗转相除法(欧几里德算法)

例如

|  |
| --- |
| (319, 377) = 29  // 错误，授权保证前一个数大于后一个数  319 % 377 = 319  // 正确  377 % 319 = 58  (319, 58)  319 % 58 = 29  (58, 29)  58 % 29 = 0 |

JAVA

|  |
| --- |
| import java.util.Scanner;  /\*\*  \* @description: 求最小公倍数  \* @author: lingjian  \* @create: 2020/7/30 9:19  \*/  public class Main {  /\*\*  \* a \* b = (a, b) \* [a, b] (a,b) 是求两个数的最大公约数 辗转相除法：大的数取余小的数，直到取余为0，那个数就是最大公约数 [a,b] 是求两个数的最小公倍数  \*/  public static void main(String[] args) {  // 输入两个整数  Scanner sc = new Scanner(System.in);  int a = sc.nextInt();  int b = sc.nextInt();  // 求出两个数的最大公约数  int max = max(a, b);  // 求出两个数的最小公倍数  int result = a \* b / max;  // 输出两个整数的最小公倍数  System.out.println(result);  }  public static int max(int a, int b) {  if (a < b) {  int temp = a;  a = b;  b = temp;  }  while (b != 0) {  int k = b;  b = a % b;  a = k;  }  return a;  }  } |

# 求解立方根

题目描述

计算一个数字的立方根，不使用库函数

详细描述：

•接口说明

原型：

public static double getCubeRoot(double input)

输入:double 待求解参数

返回值:double  输入参数的立方根，保留一位小数

## 输入描述:

待求解参数 double类型

## 输出描述:

输入参数的立方根 也是double类型

**示例1**

## 输入

216

## 输出

6.0

解题思路：

使用牛顿迭代法求解平方根，立方根

求解平方根

|  |
| --- |
| f(x) = x ^ 2 – a，那么求a的平方根等价于求f(x) = 0，由牛顿迭代公式有  x = x0 – f(x0) / f`(x0) f`(x)为函数f(x)的一阶导数f`(x0) != 0  进行迭代  x1 = x0 – f(x0) / f`(x0)  x2 = x1 – f(x1) / f`(x1)  x3 = x2 – f(x2) / f`(x2)  ……  xk+1 = xk – f(xk) / f`(xk) (k = 0,1,2,3….)  直到f(x) = 0 为止 |

求解立方根

|  |
| --- |
| f(x) = x ^ 3 – a，那么求a的立方根等价于求f(x) = 0，由牛顿迭代公式有  x = x0 – f(x0) / f`(x0) f`(x)为函数f(x)的一阶导数f`(x0) != 0  进行迭代  x1 = x0 – f(x0) / f`(x0)  x2 = x1 – f(x1) / f`(x1)  x3 = x2 – f(x2) / f`(x2)  ……  xk+1 = xk – f(xk) / f`(xk) (k = 0,1,2,3….)  直到f(x) = 0 为止 |

JAVA

|  |
| --- |
| import java.math.BigDecimal; import java.math.RoundingMode; import java.util.Scanner;  /\*\*  \* @description: 求解平方根，立方根  \* @author: lingjian  \* @create: 2020/7/30 10:03  \*/ public class SpuareCubeRoot {   /\*\*  \* 求平方根、立方根 使用牛顿迭代法 xk+1 = xk - f(xk)/f`(xk) k = 0,1,2,3... 平方根 f(xk) = x^2 - a f`(xk) = 2 \* x 立方根  \* f(xk) = x^3 - a f`(xk) = 3 \* x  \*/  public static void main(String[] args) {  Scanner sc = new Scanner(System.in);  System.out.println("请输入一个数");  double input = sc.nextDouble();   // 求平方根  double square = getSquareRoot(input);  double newSquare = new BigDecimal(square).setScale(1, RoundingMode.HALF\_UP).doubleValue();  System.out.println("平方根是" + newSquare);  // 求立方根  double cube = getCubeRoot(input);  double newCube = new BigDecimal(cube).setScale(1, RoundingMode.HALF\_UP).doubleValue();  System.out.println("立方根是" + newCube);  }   public static double getSquareRoot(double input) {  double x = 1;  double x1 = x - (x \* x - input) / (2 \* x);  // double有精度损失  while (x - x1 > 0.00000001 || x - x1 < -0.00000001) {  x = x1;  x1 = x - (x \* x - input) / (2 \* x);  }  return x1;  }   public static double getCubeRoot(double input) {  double x = 1;  double x1 = x - (x \* x \* x - input) / (3 \* x \* x);  // double有精度损失  while (x - x1 > 0.00000001 || x - x1 < -0.00000001) {  x = x1;  x1 = x - (x \* x \* x - input) / (3 \* x \* x);  }  return x1;  } } |

# 字符逆序

题目描述

将一个字符串str的内容颠倒过来，并输出。str的长度不超过100个字符。 如：输入“I am a student”，输出“tneduts a ma I”。

输入参数：

inputString：输入的字符串

返回值：

输出转换好的逆序字符串

## 输入描述:

输入一个字符串，可以有空格

## 输出描述:

输出逆序的字符串

**示例1**

## 输入

I am a student

## 输出

tneduts a ma I

解题思路

定义两个字符串a和b

a：存放正序的字符串

b：存放逆序的字符串

c：字符数组

通过String的toCharArray()方法，将字符串转换为字符数组，然后利用循环将字符数组逆序赋值给b

JAVA

|  |
| --- |
| import java.util.Scanner;  public class Main {  /\*\* 通过toCharArray()方法将字符串转换为字符数组，将字符数组逆序，在转为字符串输出即可 \*/  public static void main(String[] args) {  Scanner sc = new Scanner(System.in);  String input = sc.nextLine();  char[] chars = input.toCharArray();  char[] reverse = reverse(chars);  System.out.println(String.valueOf(reverse));  }  public static char[] reverse(char[] input) {  char[] result = new char[input.length];  for (int i = 0; i < input.length; i++) {  result[i] = input[input.length - 1 - i];  }  return result;  }  } |

# 记负均正Ⅱ

题目描述

从输入任意个整型数，统计其中的负数个数并求所有非负数的平均值，结果保留一位小数，如果没有非负数，则平均值为0

本题有多组输入数据，输入到文件末尾，请使用while(cin>>)读入

数据范围小于1e6

## 输入描述:

输入任意个整数

## 输出描述:

输出负数个数以及所有非负数的平均值

**示例1**

## 输入

-13

-4

-7

## 输出

3

0.0

解题思路

输入任意个整型数：使用Scanner的hasNextInt()方法一直输入整数型数字

在外面定义三个变量，记录负数的个数变量，记录正数的个数变量，记录正数的总和

在Scanner的hasNextInt()循环中直接判断输入的参数是否大于0，然后计算各自的总数，最后结果输出即可

JAVA

|  |
| --- |
| import java.util.Scanner;  public class Main {  public static void main(String[] args) {  Scanner sc = new Scanner(System.in);  // 记录负数的个数  int count1 = 0;  // 记录正数的个数  int count2 = 0;  // 记录正数的总和  double sum = 0.0;  // 记录正数的平均值  double avg = 0.0;  while (sc.hasNextInt()) {  int input = sc.nextInt();  if (input < 0) {  count1++;  } else {  count2++;  sum += input;  }  }  // 判断如果没有非负数，平均值为0.0  if (count2 > 0) {  avg = sum / count2;  }  System.out.println(count1);  System.out.printf("%.1f\n", avg);  }  } |

# 字符串分割

题目描述

连续输入字符串(输出次数为N,字符串长度小于100)，请按长度为8拆分每个字符串后输出到新的字符串数组，

长度不是8整数倍的字符串请在后面补数字0，空字符串不处理。

首先输入一个整数，为要输入的字符串个数。

例如：

输入：2

abc

12345789

输出：abc00000

12345678

90000000

接口函数设计如下:

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
功能:存储输入的字符串

输入:字符串

输出:无  
  
返回:0表示成功,其它返回-1  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int  AddString(char \*strValue);  
/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
功能:获取补位后的二维数组的长度

输入:无

输出:无  
  
返回:二维数组长度  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int  GetLength();

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
功能:将补位后的二维数组，与输入的二维数组做比较

输入:strInput:输入二维数组,iLen：输入的二维数组的长度

输出:无  
  
返回:若相等,返回0;不相等,返回-1.其它:-1;  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  
int  ArrCmp(char strInput[][9],int iLen);

## 输入描述:

首先输入数字n，表示要输入多少个字符串。连续输入字符串(输出次数为N,字符串长度小于100)。

## 输出描述:

按长度为8拆分每个字符串后输出到新的字符串数组，长度不是8整数倍的字符串请在后面补数字0，空字符串不处理。

**示例1**

## 输入

2

abc

123456789

## 输出

abc00000

12345678

90000000

解题思路

当字符串输入时判断字符串的长度取余8是否等于0，不等于0就在后面加7个0，接下来判断字符串长度是否大于8，大于就切分字符串，切到前8位，将切割后的字符串赋值给原字符串，继续循环判断，最后输出即可

JAVA

|  |
| --- |
| import java.util.ArrayList;  import java.util.Arrays;  import java.util.List;  import java.util.Scanner;  public class Main {  public static void main(String[] args) {  Scanner sc = new Scanner(System.in);  while (sc.hasNext()) {  int count = Integer.parseInt(sc.nextLine());  List<String> result = new ArrayList<String>();  for (int i = 0; i < count; i++) {  String s = sc.nextLine();  result.addAll(temp(s));  }  print(result);  }  }  public static List<String> temp(String s) {  List<String> result = new ArrayList<String>();  if (s.length() % 8 != 0) {  s += "0000000";  }  // 判断长度大于8  while (s.length() >= 8) {  result.add(s.substring(0, 8));  s = s.substring(8);  }  return result;  }  public static void print(List<String> list) {  for (String s : list) {  System.out.println(s);  }  }  } |

# Redraiment的走法

题目描述

题目描述

   Redraiment是走梅花桩的高手。Redraiment总是起点不限，从前到后，往高的桩子走，但走的步数最多，不知道为什么？你能替Redraiment研究他最多走的步数吗？

样例输入

6

2 5 1 5 4 5

样例输出

3

提示

Example:   
6个点的高度各为 2 5 1 5 4 5   
如从第1格开始走,最多为3步, 2 4 5   
从第2格开始走,最多只有1步,5   
而从第3格开始走最多有3步,1 4 5   
从第5格开始走最多有2步,4 5

所以这个结果是3。

接口说明

方法原型：

    int GetResult(int num, int[] pInput, List  pResult);

输入参数：  
   int num：整数，表示数组元素的个数（保证有效）。  
   int[] pInput: 数组，存放输入的数字。

输出参数：  
   List pResult: 保证传入一个空的List，要求把结果放入第一个位置。  
返回值：  
  正确返回1，错误返回0

## 输入描述:

输入多行，先输入数组的个数，再输入相应个数的整数

## 输出描述:

输出结果

**示例1**

## 输入

6

2

5

1

5

4

5

## 输出

3

解析思路

1. 存在一个数组data，设置一个同样大小的辅助数组dp，dp为一维数组，并将其所有值初始化为1

data[i]，dp[i]=1，表示data[i]为某个最长子序列中的第1个数字，

data[i]，dp[i]=1，表示data[i]为某个最长子序列中的第1个数字

1. 双重for循环，然后判断该数字之前的数字data[j]和data[i]的大小(j<i)，若data[j]<data[i]，则dp[i]的值就要变化了，因为它将不再是某个最长子序列的第一个数字；dp[i]=max(dp[i]，dp[j]+1)，要么dp[i]本身较大，要么在dp[j]前提下，加1
2. 根据二，比较数组dp[i]中的值，找出最大值，即会得到最长子序列的长度

|  |
| --- |
| import java.util.Scanner;  public class Main {  public static void main(String[] args) {  Scanner sc = new Scanner(System.in);  while (sc.hasNext()) {  int count = sc.nextInt();  int[] a = new int[count];  for (int i = 0; i < count; i++) {  a[i] = sc.nextInt();  }  System.out.println(redraiment(a));  }  sc.close();  }  public static int redraiment(int[] data) {  int n = data.length;  // 创建一个辅助数组  int[] dp = new int[n];  // 以data[1]为末元素的最长递增子序列长度为1  dp[0] = 1;  // 循环n-1次  for (int i = 1; i < n; i++) {  // dp[i]的最小值为1  dp[i] = 1;  // 循环i次  for (int j = 0; j < i; j++) {  if (data[j] < data[i]) {  // 更新dp[i]的值  dp[i] = Math.max(dp[i], dp[j] + 1);  }  }  }  int max = 1;  for (int i = 0; i < dp.length; i++) {  if (dp[i] > max) {  max = dp[i];  }  }  return max;  }  } |

# 字符统计

题目描述

如果统计的个数相同，则按照ASCII码由小到大排序输出 。如果有其他字符，则对这些字符不用进行统计。

实现以下接口：  
输入一个字符串，对字符中的各个英文字符，数字，空格进行统计（可反复调用）  
按照统计个数由多到少输出统计结果，如果统计的个数相同，则按照ASCII码由小到大排序输出  
清空目前的统计结果，重新统计  
调用者会保证：  
输入的字符串以‘\0’结尾。

## 输入描述:

输入一串字符。

## 输出描述:

对字符中的  
各个英文字符（大小写分开统计），数字，空格进行统计，并按照统计个数由多到少输出,如果统计的个数相同，则按照ASII码由小到大排序输出 。如果有其他字符，则对这些字符不用进行统计。

**示例1**

## 输入

aadddccddc

## 输出

dca

解题思路

1. 将字符串转换成字符数组。创建字符ASCLL码对应的整型数组，该数组长度必须大于128，字符的ASCLL码值就是该数组的下标，遍历字符数组，字符每出现一次对应ASCLL下标的整数就加一
2. 找出该整型数组的最大值
3. 在整型数组中匹配max，找出则将该整数下标对应的字符加入可变字符序列，max自减直至max为零

JAVA

|  |
| --- |
| import java.util.Scanner;  public class Main {  public static void main(String[] args) {  Scanner sc = new Scanner(System.in);  while (sc.hasNext()) {  String s = sc.nextLine();  System.out.println(statistics(s));  }  }  public static String statistics(String s) {  // 将字符串转换为字符数组  char[] str = s.toCharArray();  // 创建ascll码对应的字符串组  int[] ch = new int[129];  // 将字符对应ascll码值下标元素自增来统计数量  for (int i = 0; i < str.length; i++) {  ch[str[i]]++;  }  int max = 0;  // 找出字符数量最多的ascll码值  for (int j = 0; j < ch.length; j++) {  if (max < ch[j]) {  max = ch[j];  }  }  StringBuilder result = new StringBuilder();  // 按数量从大到小添加到字符串中  while (max != 0) {  for (int j = 0; j < ch.length; j++) {  if (ch[j] == max) {  result.append((char) j);  }  }  max--;  }  return result.toString();  }  } |