**实验项目名称** 实验2《一维数组》

**一、实验目的**

学习一维数组的用法、方法的定义和调用。

**二、实验内容**

1. （P236, 7.3）编写程序，读取1-100之间的整数，然后计算每个数出现的次数。假定输入是以0结束的。以下是程序运行示例：

|  |
| --- |
| 输入1-100之间的整数：2 5 5 4 3 23 2 0 [回车]  2 出现 2 次  3 出现 1 次  4 出现 1 次  5 出现 2 次  23 出现 1 次 |

2. （P237，7.10）编写一个方法，求出整数数组中最小元素的下标。如果这样的元素个数大于1，则返回最小下标。使用下面的方法头：

public static int indexofSmallestElement(double[] array)

编写测试程序，提示用户输入10个数字，调用这个方法，返回最小元素的下标，然后显示这个下标值。

3. （P236，7.5）编写程序， 读入10个数并显示互不相同的数（即一个数出现多次，但仅显示一次）。提示，读入一个数，如果它是一个新数，则将它存储在数组中，如果该数已经在数组中，则忽略它。输入之后，数组包含的都是不同的数。以下是运行示例：

输入10个整数：1 2 3 2 1 6 3 4 5 2

互不相同的数为：1 2 3 6 4 5

4.（P240，7.27）如果两个数组list1和list2内容相同，那么就说它们是相同的。使用下面的方法头编写一个方法，如果list1和list2是相同的，该方法就返回true：

public static boolean equal(int[ ] list1, int[ ] list2)

编写一个测试程序，提示用户输入两个整数数列，然后显示它们两个是否相同。以下是运行示例。注意输入的第一个数字表示数列中元素的个数。

**提示：可考虑使用230-231页Arrays类提供的方法进行组合调用**

|  |
| --- |
| 输入list1： 5 2 5 6 6 1  输入list2： 5 5 2 6 1 6  这两个数列是相同的 |

|  |
| --- |
| 输入list1： 5 5 5 6 6 1  输入list2： 5 2 5 6 1 6  这两个数列是不同的 |

5. (**附加题6.31 信用卡号的合法性**，**可选做**)信用卡号遵循下面的模式。一个信用卡号必须是13-16位的整数。它的开头必须是：

4，指visa卡

5，指master卡

37，指American Express卡

6，指Discovery卡

在1954年，IBM的Hans Luhn提出一种算法，该算法可以验证信用卡号的有效性。这个算法在确定输入的卡号是否正确，或者这张信用卡是否能被正确扫描是非常有用的。该方法通常被称为Luhn检测或Mod10 检测，描述如下（假设卡号是4388576018402626）

1. 从右至左对偶数位上的数字翻倍。如果数字翻倍后是一个两位数，那么就将这两位加在一起得到一位数。

2\*2 =4

2\*2=4

4\*2=8

1\*2=2

6\*2=12（1+2=3）

5\*2=10（1+0=1）

8\*2=16（1+6=7）

4\*2=8

1. 将第一步得到的所有一位数相加。

4+4+8+2+3+1+7+8=37

1. 将卡号里从右往左奇数位上所有数字相加。

6+6+0+8+0+7+8+3=38

1. 将第二步和第三步得到的结果相加。

37+38=75

1. 如果第四步得到的结果能被10整除，则卡号是合法的，否则是不合法的。

75%10 !=0

编写程序，提示用户输入一个long型整数的信用卡号码，显示这个数字是合法还是非法的。使用下面的方法设计程序：

/\*Return true if the card number is valid\*/

public static boolean isValid(long number)

/\*Get the result from step 2\*/

public static int sumOfDoubleEvenPlace(long number)

/\*Return this number if it is a single digit, otherwise return the sum of the two digits\*/

public static int getDigit(int number)

/\*Return sum of odd place digits in number\*/

public static int sumOfOddPlace(long number)

/\*Return true if the digit d is a prefix for number\*/

public static boolean prefixMatched(long number, int d)

/\*Return the number of digits in d\*/

public static int getSize(long d)

/\*Return the first k number of digits from number. If the number of digits in number is less than k, return number\*/

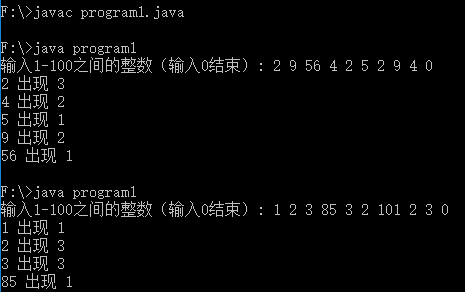
public static long getPrefix(long number, int k)

三、实验要求：

要求每个学生独立完成实验任务。

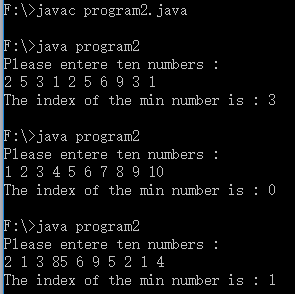
四、实验结果与分析

1. 运行结果



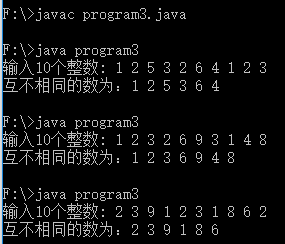
结果分析：输入1-100中n个数字以0结束，输出各个数字出现次数，运行结果正确，符合题目要求。

1. 运行结果



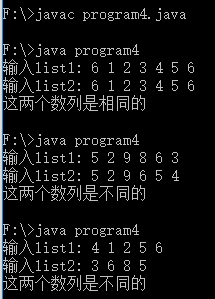
结果分析：输入10个数字，返回最小数字的最小下标，三组数据运行结果均正确，符合题目要求。

1. 运行结果



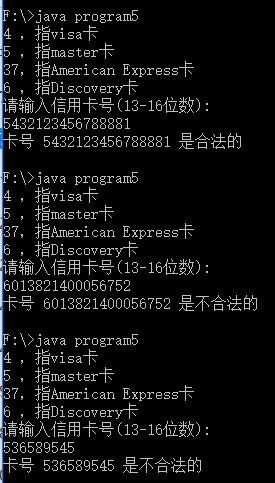
结果分析：输入10个整数，输出互不相同的数字，三组数据运行结果均正确，符合题目要求。

1. 运行结果



结果分析：输入两个数组，比较是否相同，三组数据运行结果均正确，符合题目要求。

5、运行结果



结果分析：输入13-16位的银行卡号，验证前缀及整体是否合法，六组数据运行结果均正确，符合题目要求。

五、心得体会

1. 该程序记录数组中出现的数字次数，最开始想着用两个数组，后来发现根本没必要，可以直接从1-100对比即可得知每个数字出现的次数，所以下次写程序之前一定要整理好完整的思路。但是还有一个问题就是必须设定数组大小，没办法设置为任意大小，程序有局限。
2. 该程序调用函数返回最小数的最小下标，就是函数的运用，及时更新最小数据及下标即可。
3. 该程序输出一个相异数组，可以设置循环判断语句，每次输入数据时先判断是否与之前的数据相同，不同则存入数组，同时设置count记录相异数据的个数，方便输出最后结果。
4. 该程序比较两个数组是否相同，boolean变量的运用，并且注意跳出循环条件的语句即可，对比时先判断数组长度，不等则直接返回false，在比较数据时，遇到不同数据则break跳出循环并返回false。
5. 该程序验证银行卡号的合法性，看起来比较复杂，现实上我也的确花了很多时间。不过，其实这道程序的思路是很简单的，就是先求数据位数，再取得指定奇偶位数并做对应处理，再对比数据前缀，其他就都是小问题了。

运用程序框图，先整理好思路其实这道题很快就可以做出来的，但是由于惯性思维还是经常走一步看一步，所以浪费了很多时间也打击了自己的信心。同时这也是所谓的“逐步求精”的实际简单运用，分步骤而行，简化程序的流程，化大为小。当然，这道题目还有一个很重要的点就是写函数，将每个小问题写成一个函数，彼此互不干扰最后又可以合并一起起作用。这种方式更加具有广泛性与团队开发实用性。不过就这道题而言，函数分的有点过于细微，当然作为对于练习将大的程序化简为小的程序，然后再变成一个个的简单函数的训练还是有充分的锻炼作用的。编写之后明显感觉对于程序分解、分函数书写、“逐步求精”等方面有了更加深刻的体会。

六、源程序

1.import java.util.Scanner;

public class program1 {

public static void main(String[] args) {

System.out.print("输入1-100之间的整数（输入0结束）: ");

final int ARRAYSIZE = 100; //定义常量表数组大小

Scanner input = new Scanner(System.in);

int[] list = new int[ARRAYSIZE]; //存储输入数字的数组

int number; //number表1-100的数字

int count = 0; //count用于记录数组中每个数字出现的次数

for(int i = 0; i<ARRAYSIZE; i++)

{

number = input.nextInt();

if(number!=0) //输入数字不为0程序继续

{

if(number>=0 && number<=100)

//数字在1-100之间，存入数组

list[i] = number;

}

else

break;

}

for(int i = 0; i<ARRAYSIZE+1; i++)

//输出1-100中，数组出现的数字及其次数，此循环控制数字

{

number = i + 1;

for(int j = 0; j<ARRAYSIZE; j++)

//此循环表数组中出现过的数字并统计其次数

{

if(list[j]!=0)

{

if(list[j]==number)

count++; //该数字出现次数

}

}

if(count!=0) //出现次数不为零，打印结果

System.out.print(number + " 出现 " + count +"\n");

count=0;

}

}

}

2.import java.util.Scanner;

public class program2 {

public static void main(String[] args) {

final int ARRAYSIZE = 10; //定义数组大小

double[] array = new double[ARRAYSIZE];

Scanner input = new Scanner(System.in);

System.out.println("Please entere ten numbers :");

for(int i=0; i<ARRAYSIZE; i++) //输入数组的各个元素

array[i] = input.nextDouble();

int index = indexofSmallestElement(array);

//调用函数，返回数组最小值下标

System.out.println("The index of the min number is : " + index);

}

public static int indexofSmallestElement(double[] array){

int indexOfMin = 0; //最小数下标

double min = array[0]; //数组中的最小数

for(int i = 1; i<array.length; i++)

{

if(array[i]<min)

//如果数组中的数小，更新min及indexOfMin

{

min = array[i];

indexOfMin = i;

}

}

return indexOfMin;

}

}

3.import java.util.Scanner;

public class program3 {

public static void main(String[] args) {

Scanner input = new Scanner(System.in);

final int ARRAYSIZE = 10; //定义数组大小

int[] array = new int[ARRAYSIZE];

int number;

int count=0; //count表示实际数组不重复数字个数

System.out.print("输入10个整数: ");

array[0] = input.nextInt();

for(int i = 1; i<array.length; i++)

{

number = input.nextInt();

for(int j = 0; j<i; j++)

//判断输入的数字与前面数组中的数字是否相等

{

if(array[j]==number)

break;

else

{

if(j==(i-1)) //输入的数字与前面的数字不同

{

count++; //数组下标加一

array[count]=number;

}

}

}

}

System.out.print("互不相同的数为：");

for(int i=0; i<=count; i++)

System.out.print(array[i] + " ");

System.out.println();

}

}

4.import java.util.Scanner;

public class program4 {

public static void main(String[] args) {

Scanner input = new Scanner(System.in);

//数组1--大小及数据输入

System.out.print("输入list1: ");

int arraysize1 = input.nextInt(); //数组1大小

int[] list1 = new int[arraysize1];

for(int i=0; i<list1.length; i++) //数组1数据输入

list1[i] = input.nextInt();

//数组2--大小及数据输入

System.out.print("输入list2: ");

int arraysize2 = input.nextInt(); //数组2大小

int[] list2 = new int[arraysize2];

for(int i=0; i<list2.length; i++) //数组2数据输入

list2[i] = input.nextInt();

if(equal(list1, list2)) //调用函数判断两个数组是否相同

System.out.println("这两个数列是相同的");

else

System.out.println("这两个数列是不同的");

}

public static boolean equal(int[] list1, int[] list2)

{

if(list1.length!=list2.length) //数组长度不等，直接返回false

{

return false;

}

else //数组长度相等，比较具体数据

{

boolean compare = false;

for(int i=0; i<list1.length; i++)

{

if(list1[i]==list2[i]) //数组数据相等，布尔变量为真

compare = true;

else //数组数据不相等，布尔变量为假，同时跳出循环判断

{

compare = false;

break;

}

}

return compare;

}

}

}

5.import java.util.Scanner;

public class program5 {

public static void main(String[] args) {

Scanner input = new Scanner(System.in);

System.out.println("4 ，指visa卡\n" + "5 ，指master卡\n" +

"37，指American Express卡\n" + "6 ，指Discovery卡\n"

+ "请输入信用卡号(13-16位数):");

long cardNumber = input.nextLong();

if(isValid(cardNumber)) //调用卡号验证函数，返回布尔变量

System.out.println("卡号 " + cardNumber +" 是合法的");

else

System.out.println("卡号 " + cardNumber + " 是不合法的");

}

/\*Return true if the card number is valid\*/

public static boolean isValid(long number)

//卡号验证的主要步骤函数

{

boolean valid = false;

if(number<1E13 || number>=1E17)

//输入的卡号位数不对，卡号验证不通过

return valid;

else

{

if(prefixMatched(number, 37) || prefixMatched(number, 4) || prefixMatched(number, 5) || prefixMatched(number, 6))

//调用前缀匹配函数，前缀不对直接返回false

{

//验证第第五步，奇偶位数之和能否整除10

if( (sumOfDoubleEvenPlace(number) + sumOfOddPlace(number) ) % 10 == 0 )

valid = true;

else

valid = false;

}

}

return valid;

}

/\*Get the result from step 2\*/

public static int sumOfDoubleEvenPlace(long number)

//求偶数位数字的两倍之和

{

int sumOfEven = 0;

int digit;

for(int i = 2; i<=getSize(number); i += 2 )

//i表示从右边第二位开始计算

{

digit = (int)getPrefix(number, i);

//getPrefix函数返回number中从右至左的i位数

sumOfEven += getDigit(digit);

//getDigit函数确定偶数位两倍是一位数还是两位数并返回一位数

}

return sumOfEven;

}

/\*Return this number if it is a single digit, otherwise return the sum of the two digits\*/

//从右至左偶数位两倍一位数则直接返回，两位数则相加返回一位数

public static int getDigit(int number)

{

int temp = number \* 2;

if( temp > 9 )

number = temp/10 + temp%10;

//两位数则个位加十位成一位数

else

number = temp;

return number;

}

/\*Return sum of odd place digits in number\*/

//奇数位所有数字求和

public static int sumOfOddPlace(long number)

{

int sumOfOdd = 0;

for(int i = 1; i<=getSize(number); i += 2 )

//i表示从右边第一位开始计算

sumOfOdd += (int)getPrefix(number, i);

//getPrefix函数返回number中从右至左第i位数

return sumOfOdd;

}

/\*Return true if the digit d is a prefix for number\*/

//此函数验证卡号前缀是否符合要求，d表示标准前缀

public static boolean prefixMatched(long number, int d)

{

boolean prefixMatched = false;

int prefix = (int)(number / (Math.pow((int)10, (int)( getSize(number)-getSize(d) ))));

if(d==prefix)

prefixMatched = true;

return prefixMatched;

}

/\*Return the number of digits in d\*/

//求long数据的位数

public static int getSize(long d)

{

int a = 1;

long sh = 10; //控制number从右向左一位一位的移动

while(d/sh!=0) //判断number是否到达最左边结束

{

a++; //记录数据的位数

sh \*= 10; //控制数位左移

}

return a;

}

/\*Return the first k number of digits from number. If the number of digits in number is less than k, return number\*/

//获得数据number从右至左的k位数

public static long getPrefix(long number, int k)

{

long prefix;

if(k<=getSize(number))

prefix = (int)( ( number % (Math.pow((int)10, (int)k)) ) / ( Math.pow((int)10, (int)k-1) ));

//调用math.pow函数先取余再除得该数位的数

else //k大于number的位数

prefix = number;

return prefix;

}

}