

# 实验7

## 练习1

### 题目

求一元  $n$  次多项式  $P_n(x)$  的导函数。

说明：

(1)  $P_n(x) = c_1x^{e_1} + c_2x^{e_2} + \cdots + c_mx^{e_m}$ ，其中  $c_i$  (实数) 是指数为  $e_i$  (非负整数) 的项的非零系数， $0 \leq e_1 < e_2 < \cdots < e_m = n$ 。

(2) 多项式和结果多项式都用带头结点的单链表实现，其结点包含 3 个域：系数域 coef，指数域 expn 和指针域 next。

(3) 输出部分要考虑结果为 0 的情况。

### 解析

定义Node类型，包含coef(double)、expn(int)及next(Node)三个变量。

定义Polynomial类型（实为单链表），采用和实验6练习1中类似的实现，但简化了很多。

Polynomial的构造方法需要输入两个相同长度的数组coefs和expns，类型分别为double[]和int[]，将通过这两个数组构造单链表。除length()、append()、insert()、remove()这些基本方法外，Polynomial还重写了toString方法。

在输入coefs和expns后，使用这两个数组创建Polynomial单链表poly，并对每一个结点求导，最后得到的coefs和expns即为所求值。

### 代码

```
import java.util.Scanner;

class Node {
    public double coef;
    public int expn;
    public Node next;

    public Node() {
        this(0, 0, null);
    }

    public Node(double coef, int expn) {
        this(coef, expn, null);
    }

    public Node(double coef, int expn, Node next) {
        this.coef = coef;
        this.expn = expn;
        this.next = next;
    }
}

class Polynomial {
```

```

private Node head;
private int length;

public Polynomial() {
    length = 0;
    head = new Node();
}

public Polynomial(double[] coefs, int[] expns) throws Exception {
    this();
    if (coefs.length != expns.length) {
        throw new Exception("The length of coefs must be equal to that of
expns.");
    }
    this.length = coefs.length;
    Node p = head;
    for (int i = 0; i < this.length; i++) {
        p.next = new Node(coefs[i], expns[i]);
        p = p.next;
    }
}

public Node head() {
    return this.head;
}

public int length() {
    return this.length;
}

public void append(double coef, int expn) {
    Node p = head;
    for (int i = 0; i < length; i++) {
        p = p.next;
    }
    p.next = new Node(coef, expn);
    length++;
}

public void insert(int index, double coef, int expn) throws Exception {
    if (index < 0 || index > length) {
        throw new Exception("Invalid index " + index);
    }
    Node p = head;
    for (int i = 0; i < index; i++) {
        p = p.next;
    }
    p.next = new Node(coef, expn, p.next);
    length++;
}

public void remove(int index) throws Exception {
    if (index < 0 || index >= length) {
        throw new Exception("Invalid index " + index);
    }
    Node p = head;
    for (int i = 0; i < index; i++) {
        p = p.next;
    }
}

```

```

    }
    p.next = p.next.next;
    length--;
}

public String toString() {
    // coef
    Node p = head.next;
    StringBuilder builder = new StringBuilder();
    builder.append("coef: [");
    if (length > 0) {
        builder.append(p.coef);
    }
    for (int i = 1; i < length; i++) {
        p = p.next;
        builder.append(", ");
        builder.append(p.coef);
    }
    builder.append("]");
    // expn
    p = head.next;
    builder.append("\nexpn: [");
    if (length > 0) {
        builder.append(p.expn);
    }
    for (int i = 1; i < length; i++) {
        p = p.next;
        builder.append(", ");
        builder.append(p.expn);
    }
    builder.append("]");
    return builder.toString();
}
}

public class Exercise_1 {
    public static void main(String[] args) throws Exception {
        Scanner in = new Scanner(System.in);
        // get term number
        System.out.println("Input n:");
        int n = in.nextInt();
        // input coefs
        System.out.println("Input all coef:");
        double[] coefs = new double[n];
        for (int i = 0; i < n; i++) {
            coefs[i] = in.nextDouble();
        }
        // input expns
        System.out.println("Input all expn:");
        int[] expns = new int[n];
        for (int i = 0; i < n; i++) {
            expns[i] = in.nextInt();
        }
        in.close();
        // create poly and calculate
        Polynomial poly = new Polynomial(coefs, expns);
        Node p = poly.head();
        for (int i = 0; i < n; i++) {

```

```

        p = p.next;
        if (p.expn == 0) {
            p.coef = 0;
        } else {
            p.coef *= p.expn;
            p.expn -= 1;
        }
    }
    System.out.println(poly);
}
}

```

## 输入

```

Input n:
3
Input all coef:
1 2 3
Input all expn:
1 2 3

```

## 输出

```

coef: [1.0, 4.0, 9.0]
expn: [0, 1, 2]

```

## 练习2

### 题目

有编号依次为 1 至 10 的 10 个人各拿一只水桶同时来到一只水龙头前打水，水龙头注满各水桶所需的时间依次为 60, 30, 80, 20, 90, 40, 100, 10, 70, 50 秒。对这 10 个人进行排队，让他们所花时间的总和（包括每人等待和接水所花时间）最小。输出排队后的编号次序和该时间总和。

### 解析

易证按照打水所需时间从小到大依次排列即可。

定义函数min\_index，接受一个int数组作为输入，输出数组中最小元素的下标。

每次调用min\_index找出输入数组costTime中最小值的下标加入数组order，并将找到的最小值更改为 $\infty$ （用0x7FFFFFFF表示）。当所有值都被改为 $\infty$ 后，order即为所求结果。根据order计算总时间即可。

### 代码

```

public class Exercise_2 {
    public static int min_index(int[] arr) {
        int min = arr[0];
        int minIndex = 0;
        for (int i = 1; i < arr.length; i++) {
            if (arr[i] < min) {
                min = arr[i];
            }
        }
        return minIndex;
    }
}

```

```

        minIndex = i;
    }
}
return minIndex;
}

public static void main(String[] args) throws Exception {
    int[] costTime = { ... }; // costTime的具体值将在“输入”中给出
    // get order
    int[] costTimeTmp = costTime.clone();
    int[] order = new int[costTime.length];
    for (int i = 0; i < costTime.length; i++) {
        int minIndex = min_index(costTimeTmp);
        order[i] = minIndex;
        costTimeTmp[minIndex] = 0x7FFFFFFF;
    }
    // print order
    System.out.print("Order: [" + order[0]);
    for (int i = 1; i < order.length; i++) {
        System.out.print(", " + order[i]);
    }
    System.out.println("]");
    // print total time
    int totalTime = 0;
    for (int i = 0; i < order.length; i++) {
        totalTime += costTime[order[i]] * (order.length - i);
    }
    System.out.println("Total Time: " + totalTime + "s");
}
}

```

## 输入

```
{60, 30, 80, 20, 90, 40, 100, 10, 70, 50}
```

## 输出

```
Order: [7, 3, 1, 5, 9, 0, 8, 2, 4, 6]
Total Time: 2200s
```

## 练习3

### 题目

下面两个算式中的每个汉字都代表 0 至 9 中的数字（相同的汉字代表相同的数字，不同的汉字代表不同的数字）。破译这两个算式。

年年×岁岁=花相似

岁岁÷年年=人÷不同

## 解析

定义函数has\_repeated，输入数组arr、起始下标start和结束下标end，查找数组元素是否重复。

定义int数组nums，含八个元素，按下标从0到7分别表示“年”、“岁”、“花”、“相”、“似”、“人”、“不”、“同”。

由题意可知，年年必定大于岁岁，且“岁”只能取1或2 ( $22 \times 33 = 726$ ,  $33 \times 44 = 1452 > 999$ )

然后使用暴力搜索查找结果，需要使用三重循环。

## 代码

```
import java.util.Arrays;

public class Exercise_3 {
    // 检查数组元素是否有重复
    public static boolean has_repeated(int[] arr, int start, int end) {
        int length = end - start;
        for (int i = start; i < length - 1; i++) {
            for (int j = i + 1; j < length - 1; j++) {
                if (arr[i] == arr[j]) {
                    return true;
                }
            }
        }
        return false;
    }

    public static void main(String[] args) throws Exception {
        int[] nums = new int[8];
        for (nums[1] = 1; nums[1] <= 2; nums[1]++) { // 22*33=726,
            33*44=1452>999
            for (nums[0] = nums[1] + 1; nums[0] <= 9; nums[0]++) { // 年年必定大于
                岁岁

                int nn = 10 * nums[0] + nums[0];
                int ss = 10 * nums[1] + nums[1];
                int tmp = nn * ss;
                if (tmp > 999) {
                    break;
                }
                nums[2] = tmp / 100; // 花
                nums[3] = tmp % 100 / 10; // 相
                nums[4] = tmp % 10; // 似
                if (!has_repeated(nums, 0, 5)) {
                    for (nums[5] = 1; nums[5] <= 9; nums[5]++) { // 人
                        if (!has_repeated(nums, 0, 6)) {
                            if ((nn * nums[5]) % ss == 0) {
                                tmp = (nn * nums[5]) / ss;
                                if (tmp < 10 || tmp > 99) {
                                    continue;
                                }
                                nums[6] = tmp / 10; // 不
                                nums[7] = tmp % 10; // 同
                                if (!has_repeated(nums, 0, 8)) {
                                    System.out.println(Arrays.toString(nums));
                                }
                            }
                        }
                    }
                }
            }
        }
    }
}
```

```
}
    }
  }
}
```

## 输入

无

## 输出

```
[4, 2, 9, 6, 8, 5, 1, 0]
[4, 2, 9, 6, 8, 7, 1, 4]
```

## 练习4

### 题目

从 18, 19, 12, 17, 20, 11, 8, 15, 16, 7 中找出所有两数之和为质数（素数）的数对，如（18, 19）。

### 解析

定义函数is\_prime，接受一个整数，返回一个布尔量表示是否是素数。

使用二重循环暴力搜索输入数组nums，打印所有数对。

### 代码

```
public class Exercise_4 {
    // 判断是否n为素数
    public static boolean is_prime(int n) {
        for (int i = 2; i <= n / 2; i++) {
            if (n % i == 0) {
                return false;
            }
        }
        return true;
    }

    public static void main(String[] args) throws Exception {
        int[] nums = { ... }; // nums的具体值将在“输入”中给出
        for (int i = 0; i < nums.length; i++) {
            for (int j = i + 1; j < nums.length; j++) {
                if (is_prime(nums[i] + nums[j])) {
                    System.out.println("(" + nums[i] + ", " + nums[j] + ")");
                }
            }
        }
    }
}
```

## 输入

```
{18, 19, 12, 17, 20, 11, 8, 15, 16, 17}
```

## 输出

```
(18, 19)
(18, 11)
(19, 12)
(12, 17)
(12, 11)
(12, 17)
(17, 20)
(20, 11)
(20, 17)
(11, 8)
(8, 15)
(15, 16)
```

## 练习5

### 题目

求 10 个最小的连续自然数，它们都是合数。

### 解析

原理比较简单，不过多说明。

### 代码

```
import java.util.Arrays;

public class Exercise_5 {
    // 判断n是否为素数
    public static boolean is_prime(int n) {
        for (int i = 2; i <= n / 2; i++) {
            if (n % i == 0) {
                return false;
            }
        }
        return true;
    }

    public static void main(String[] args) throws Exception {
        int count = 0;
        int n = 1;
        int[] nums = new int[10];
        while (count < 10) {
            n += 1;
            if (is_prime(n)) {
                count = 0;
            } else {
                nums[count] = n;
                count++;
            }
        }
        Arrays.sort(nums);
        for (int i = 0; i < 10; i++) {
            System.out.print(nums[i] + " ");
            if (i % 10 == 9) {
                System.out.println();
            }
        }
    }
}
```



```
        nums[count] = n;  
        count += 1;  
    }  
}  
System.out.println(Arrays.toString(nums));  
}
```

## 输入

无

## 输出

[114, 115, 116, 117, 118, 119, 120, 121, 122, 123]

## 心得体会

1. 将单链表稍加修改可以简洁高效地解决许多问题。
2. 解决问题前进行适当的分析可以大大提升算法效率。