实验项目一：线性结构

（一） 实验目的

1、理解线性表的顺序存储结构；

2、熟练掌握顺序表结构及其有关算法的设计；

3、理解线性表的链式存储结构；

4、熟练掌握动态链表结构及其有关算法的设计；

5、根据具体问题的需要，设计出合理的表示数据的链表结构，并设计相关算法；

6、深入了线性表的特性，在实际问题背景下灵活线性表解决问题。

（二）实验内容

1、以顺序表和链表作存储结构，实现线性表的插入、删除；并通过测试数据验证。

（1）有一个整数顺序表，设计一个尽可能高效的算法，删除所有值为x的结点。并给出算法的时间复杂度和空间复杂度。例如：L={1, 2, 3, 4, 2, 6, 2, 7}，x =2，删除后：L={1，3, 4, 6, 7}。

```java
public class SequentialList {
    private int[] data; // 存储数据的数组
    private int size; // 当前元素个数

    public SequentialList(int capacity) {
        data = new int[capacity];
        size = 0;
    }

    // 添加元素
    public void add(int element) {
        if (size < data.length) {
            data[size++] = element;
        }
    }

    // 删除所有值为x的元素
    // 时间复杂度：O(n)，只需要遍历一次数组
    // 空间复杂度：O(1)，只使用了常数个额外变量
    public void removeAll(int x) {
        int k = 0; // k指向新数组的位置

        // 遍历数组，将不等于x的元素移到前面
        for (int i = 0; i < size; i++) {
            if (data[i] != x) {
                data[k] = data[i];
                k++;
            }
        }

        // 更新size为新的长度
        size = k;
    }
```

```
34
35      // 获取当前元素个数
36      public int size() {
37          return size;
38      }
39
40      // 获取指定位置的元素
41      public int get(int index) {
42          if (index >= 0 && index < size) {
43              return data[index];
44          }
45          throw new IndexOutOfBoundsException();
46      }
47
48      // 打印顺序表
49      @Override
50      public String toString() {
51          StringBuilder sb = new StringBuilder();
52          sb.append("{");
53          for (int i = 0; i < size; i++) {
54              sb.append(data[i]);
55              if (i < size - 1) {
56                  sb.append(", ");
57              }
58          }
59          sb.append("}");
60          return sb.toString();
61      }
62
63      // 测试主方法
64      public static void main(String[] args) {
65          // 创建顺序表并添加测试数据
66          SequentialList list = new SequentialList(10);
67          int[] testData = { 1, 2, 3, 4, 2, 6, 2, 7 };
68          for (int num : testData) {
69              list.add(num);
70          }
71
72          System.out.println("原始顺序表: " + list);
73
74          // 删除所有值为2的元素
75          list.removeAll(2);
76          System.out.println("删除值为2后的顺序表: " + list);
77      }
78  }
```

```
C:\programs\jdk-21.0.4\bin\java.exe "-javaagent:D:\Idea\IntelliJ IDEA 2024.2.3\lib\idea_rt.jar=60761:D:\
原始顺序表: {1, 2, 3, 4, 2, 6, 2, 7}
删除值为2后的顺序表: {1, 3, 4, 6, 7}
```

（2）有一个整数单链表，设计一个尽可能高效的算法，将所有的负整数的元素移动到其他元素前面。例如：L={1, 2 ,-1，-2, 3，-3, 4}，x =2，移动后：L={-1，-2，-3,1, 2, 3, 4}。

```
1   public class LinkedList {
2       // 定义节点类
```

```
    private static class Node {
        int data;
        Node next;

        Node(int data) {
            this.data = data;
            this.next = null;
        }
    }

    private Node head; // 头节点

    public LinkedList() {
        head = null;
    }

    // 在链表末尾添加节点
    public void add(int data) {
        Node newNode = new Node(data);
        if (head == null) {
            head = newNode;
            return;
        }
        Node current = head;
        while (current.next != null) {
            current = current.next;
        }
        current.next = newNode;
    }

    // 将所有负数移动到非负数前面
    // 时间复杂度：O(n)，只需要遍历一次链表
    // 空间复杂度：O(1)，只使用了常数个额外变量
    public void moveNegativesToFront() {
        if (head == null || head.next == null) {
            return; // 空链表或只有一个节点，无需移动
        }

        Node negativeEnd = null; // 负数区域的末尾节点
        Node current = head;
        Node prev = null;

        // 找到第一个负数，作为新的头节点
        while (current != null && current.data >= 0) {
            prev = current;
            current = current.next;
        }

        // 如果没有找到负数，直接返回
        if (current == null) {
            return;
        }

        // 将第一个负数移到头部
        if (prev != null) {
            prev.next = current.next;
```

```
            current.next = head;
            head = current;
        }
        negativeEnd = head;

        // 继续遍历剩余节点
        current = (prev == null) ? current.next : prev.next;
        prev = (prev == null) ? head : prev;

        while (current != null) {
            if (current.data < 0) {
                // 将负数节点移动到负数区域的末尾
                prev.next = current.next;
                current.next = negativeEnd.next;
                negativeEnd.next = current;
                negativeEnd = current;
                current = prev.next;
            } else {
                prev = current;
                current = current.next;
            }
        }
    }

    // 打印链表
    @Override
    public String toString() {
        StringBuilder sb = new StringBuilder();
        sb.append("{");
        Node current = head;
        while (current != null) {
            sb.append(current.data);
            if (current.next != null) {
                sb.append(", ");
            }
            current = current.next;
        }
        sb.append("}");
        return sb.toString();
    }

    // 测试主方法
    public static void main(String[] args) {
        LinkedList list = new LinkedList();

        // 添加测试数据
        int[] testData = { 1, 2, -1, -2, 3, -3, 4 };
        for (int num : testData) {
            list.add(num);
        }

        System.out.println("原始链表: " + list);

        // 移动负数到前面
        list.moveNegativesToFront();
        System.out.println("移动负数后的链表: " + list);
```

```
    }
}
```

```
C:\programs\jdk-21.0.4\bin\java.exe "-javaagent:D:\Idea\Intel
原始链表: {1, 2, -1, -2, 3, -3, 4}
移动负数后的链表: {-1, -2, -3, 1, 2, 3, 4}
```

2、分别以顺序表和单链表作存储结构，实现有序表的合并，并通过测试验证。

（1）有两个集合采用递增有序的整数顺序表A、B存储，请设计一个算法求两个集合的并集C，C仍然用顺序表存储，并给出算法的时间复杂度和空间复杂度。A={1, 3, 5, 7}，B={1，2, 4, 5, 7}，并集C={1, 2, 3, 4, 5, 7}。

```java
public class OrderedArrayMerge {
    // 合并两个有序数组，返回并集
    public static int[] merge(int[] A, int[] B) {
        if (A == null || B == null) {
            return A == null ? B : A;
        }

        // 创建临时数组存储结果
        int[] temp = new int[A.length + B.length];
        int i = 0, j = 0, k = 0;

        // 使用双指针法合并两个有序数组
        while (i < A.length && j < B.length) {
            if (A[i] < B[j]) {
                if (k == 0 || temp[k - 1] != A[i]) {
                    temp[k++] = A[i];
                }
                i++;
            } else if (A[i] > B[j]) {
                if (k == 0 || temp[k - 1] != B[j]) {
                    temp[k++] = B[j];
                }
                j++;
            } else {
                // 相等时只添加一次
                if (k == 0 || temp[k - 1] != A[i]) {
                    temp[k++] = A[i];
                }
                i++;
                j++;
            }
        }

        // 处理剩余元素
        while (i < A.length) {
            if (k == 0 || temp[k - 1] != A[i]) {
                temp[k++] = A[i];
            }
            i++;
        }
```

```
41
42          while (j < B.length) {
43              if (k == 0 || temp[k - 1] != B[j]) {
44                  temp[k++] = B[j];
45              }
46              j++;
47          }
48
49          // 创建最终结果数组
50          int[] result = new int[k];
51          System.arraycopy(temp, 0, result, 0, k);
52          return result;
53      }
54
55      public static void main(String[] args) {
56          // 测试用例
57          int[] A = { 1, 3, 5, 7 };
58          int[] B = { 1, 2, 4, 5, 7 };
59
60          System.out.println("集合A: ");
61          printArray(A);
62          System.out.println("集合B: ");
63          printArray(B);
64
65          int[] C = merge(A, B);
66          System.out.println("并集C: ");
67          printArray(C);
68      }
69
70      private static void printArray(int[] arr) {
71          for (int num : arr) {
72              System.out.print(num + " ");
73          }
74          System.out.println();
75      }
76  }
```

```
C:\programs\jdk-21.0.4\bin\java.exe "-javaagent:D:\Idea\IntelliJ IDEA 2024.2.3\lib\idea_rt.jar=62
集合A:
1 3 5 7
集合B:
1 2 4 5 7
并集C:
1 2 3 4 5 7

Process finished with exit code 0
```

（2）有两个集合采用递增有序的整数单链表A、B存储，请设计一个算法求两个集合的交集C，C仍然用单链表存储，并给出算法的时间复杂度和空间复杂度。A={1, 3, 5, 7}，B={1，2, 4, 5, 7}，并集C={1，5, 7}。

```
1  class ListNode {
2      int val;
3      ListNode next;
4
5      ListNode(int val) {
```

```
        this.val = val;
    }
}

public class OrderedLinkedListIntersect {
    // 创建有序链表
    public static ListNode createList(int[] arr) {
        if (arr == null || arr.length == 0)
            return null;

        ListNode dummy = new ListNode(0);
        ListNode current = dummy;

        for (int num : arr) {
            current.next = new ListNode(num);
            current = current.next;
        }

        return dummy.next;
    }

    // 计算两个有序链表的交集
    public static ListNode intersect(ListNode A, ListNode B) {
        if (A == null || B == null)
            return null;

        ListNode dummy = new ListNode(0);
        ListNode current = dummy;

        // 使用双指针遍历两个链表
        while (A != null && B != null) {
            if (A.val < B.val) {
                A = A.next;
            } else if (A.val > B.val) {
                B = B.next;
            } else {
                // 找到相同元素
                current.next = new ListNode(A.val);
                current = current.next;
                A = A.next;
                B = B.next;
            }
        }

        return dummy.next;
    }

    // 打印链表
    public static void printList(ListNode head) {
        ListNode current = head;
        while (current != null) {
            System.out.print(current.val + " ");
            current = current.next;
        }
        System.out.println();
    }
```

```
62
63      public static void main(String[] args) {
64          // 测试用例
65          int[] arrA = { 1, 3, 5, 7 };
66          int[] arrB = { 1, 2, 4, 5, 7 };
67
68          ListNode A = createList(arrA);
69          ListNode B = createList(arrB);
70
71          System.out.println("链表A: ");
72          printList(A);
73          System.out.println("链表B: ");
74          printList(B);
75
76          ListNode C = intersect(A, B);
77          System.out.println("交集C: ");
78          printList(C);
79      }
80  }
```

```
C:\programs\jdk-21.0.4\bin\java.exe "-javaagent:D:\Idea\Ir
链表A:
1 3 5 7
链表B:
1 2 4 5 7
交集C:
1 5 7

Process finished with exit code 0
```

（三）实验技能要求

1、理解顺序表、链表的特点，头结点、头指针的概念及设置头结点的优点；

2、掌握顺序表和链表存储数据的方法，进行数据插入、删除等相关算法的设计。

（四）实验操作要求

能根据题目要求，正确选用线性数据结构，熟练使用 C、Java等编程语言， 编程实现算法，对实际问题进行求解，并能设计测试用例，对实验结果进行测试、 分析。