利用单链表对两个多项式求和实验报告

一、问题分析

- 处理对象：两个多项式每一项的系数以及幂的阶数，以链表储存和处理
- 需要实现两个多项式相加求和，输出求和后结果
- 将结果储存在新链表中，输出新链表的节点数据
- 题目输入样例：

```
3
3 5
-2 1
4 0
4
2 3
-1 2
1 1
3 0
```

- 求解过程：样例为多项式 3x^5-2x+4 与多项式 2x^3-x^2+x+3 相加，计算结果：3x^5+2x^3-x^2-x+7。

二、数据结构和算法设计

1. 抽象数据类型设计：

采用链表 ADT，即 List ADT，其中封装了链表的多种函数接口，如链表的构造、析构、插入、末尾追加等，具体设计如下：

```
template <typename E> class List { // List ADT抽象数据库
private:
       void operator =(const List&) {}
         // Protect assignment
       List(const List&) {}
        // Protect copy constructor
public:
       List() {}
    // Default constructor
       virtual ~List() {} // Base destructor

// Clear contents from the list, to make it empty.
       virtual void clear() = 0;

  // Insert an element at the current location.
  // item: The element to be inserted
virtual void insert(const E& item1,const E& item2) = 0;

    // Append an element at the end of the list.
   // item: The element to be appended.
 virtual void append(const E& item1,const E& item2) = 0;

  // Remove and return the current element.
  // Return: the element that was removed.
       virtual E remove() = 0;

 // Set the current position to the start of the list
       virtual void moveToStart() = 0;

 // Set the current position to the end of the list
       virtual void moveToEnd() = 0;
```

```
    // Move the current position one step left.
    //   No change if already at beginning.
        virtual void prev() = 0;

    // Move the current position one step right.
    // No change if already at end.
        virtual void next() = 0;
    // Return: The number of elements in the list.
        virtual int length() const = 0;

    // Return: The position of the current element.
        virtual int currPos() const = 0;

        // Set current position.
        // pos: The position to make current.
        virtual void moveToPos(int pos) = 0;

        // Return: The current element.
        virtual const E& getValue1() const = 0;
        virtual const E& getValue2() const = 0;
};
```

2. 物理数据对象设计：

首先是模版类 Link，用于实例化链表中的节点，每个节点中公有成员 element1，elemnet2 分别储存多项式中每一项的系数及阶数，以及一个节点指针 next 用于指向下一个节点，将一个多项式的每一项连接起来。其构造函数用于初始化其私有成员；

其次式模版类 List，其公有继承了 List 抽象数据类型，具体实现其中接口函数，实现对链表的基本操作，且包含三个私有成员，其中 head、tail 作为链表头指针、尾指针，curr 用于遍历链表对链表指定进行相关操作，三者数据类型均为 Link*。

3.算法思想设计

1. 根据输入的 a（第一项的项数），b（第二项的项数）构造三个链表 llist1，llist2，llist3，分别表示第一个多项式、第二个多项式，前两个多项式相加后的新的多项式。它们中每个节点代表多项式中的一项。

2. 计算相加 llsit1 和 llist2 所代表的多项式，将每一项的结果存入 llist3，得到新的多项式。

3. 顺序输出 llist3 的每一个节点的 element1，element2。

4.关键功能的算法步骤

1. 计算 llist1 和 llist2 所代表的多项式：用 llist1 与 llist2 的 curr 指针循环遍历链表顺序比较 llist1 和 llist2 中每一项的阶数大小。

若相等则可以相加合并,储存到 llist3 中，

(if llist1.getValue1() == llist2.getValue1())

{

```
  llist3.insert(llist1.getValue1() + llist2.getValue1(),
llist1.getValue2());
```

}

不相等则先将阶数大的那一项存入 llist3，并将对应的链表 curr 指针往前移一个节点判断下一项(llist.next())，每存一项 llist3 的 curr 指针也相应往前移一个节点，当 llst1 及 llist2 的 curr 指针都在尾节点前一项时退出循环。

(while llist1.currPos < a && llist2.currPos < b)

三、算法性能分析

1. 将分别 n，m 个节点插入 llist1、llist2，构造多项式，每次执行插入操作需要时间为一个常量，记为 c1，分别执行 a，b 次，时间代价为 n*c1+m*c1（N*c1);

2. 循环判断 llist1 与 llist2 每一项阶数大小，循环内都是基本操作，时间代价为一常量，记为 c2，循环次数最少（最佳情况）为 max(n,m)，最多（最差情况）为 n+m（N），则其时间复杂度为Θ（N）.

3.优点在于没有浪费空间，且易操作更新（如访问元素、插入元素），且平均判断次数与最大判断次数相近，计算速度快；缺点在于需要排序、顺序储存（不过题目输入方式为严格降序）。