**西北大学信息学院数据结构实验课程实验报告**

**实验二 线性表及其应用**

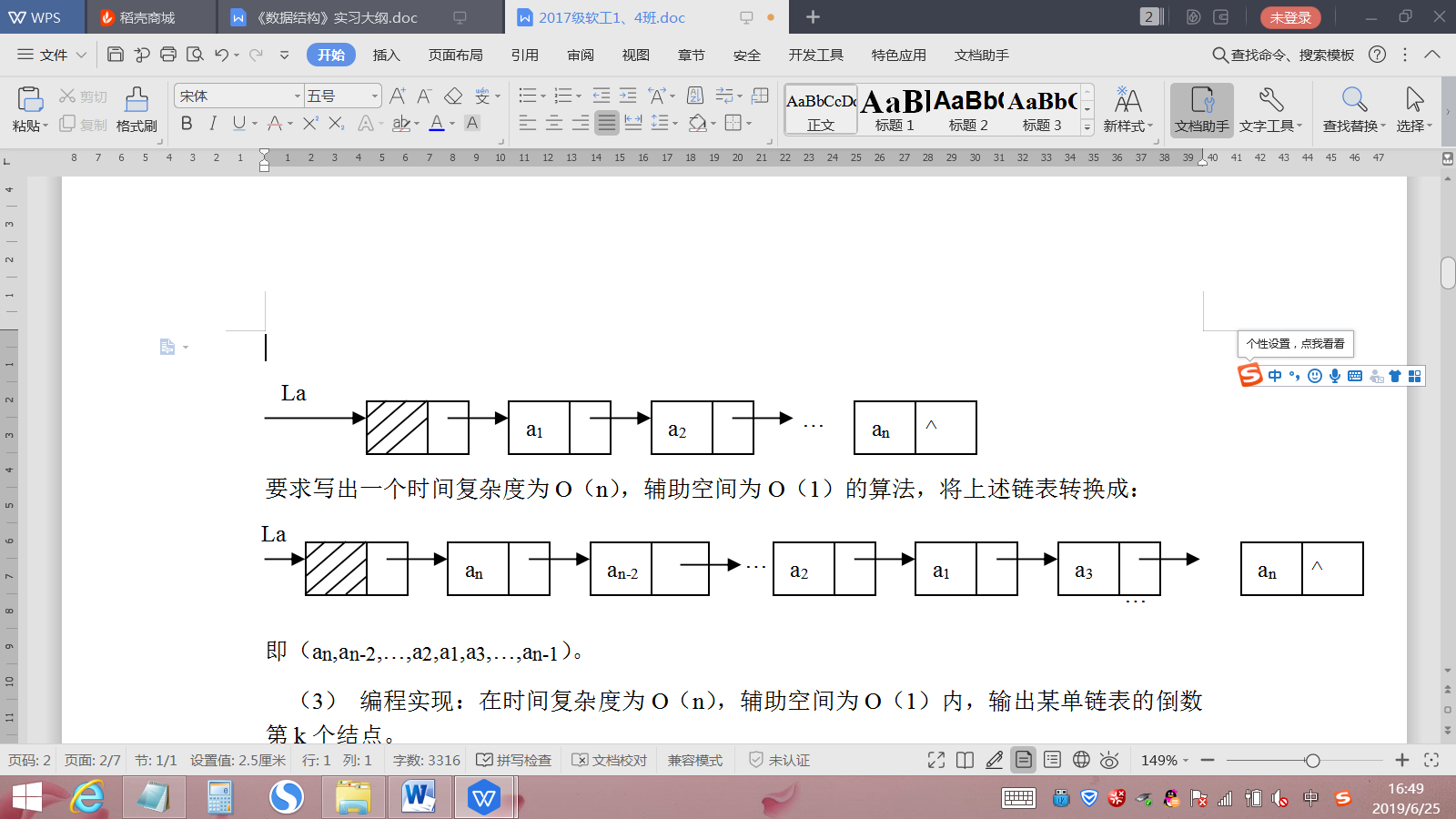
**班级：2020计科2 学号： 2020115025 姓名：薄劲阳**

**一 实验目的**

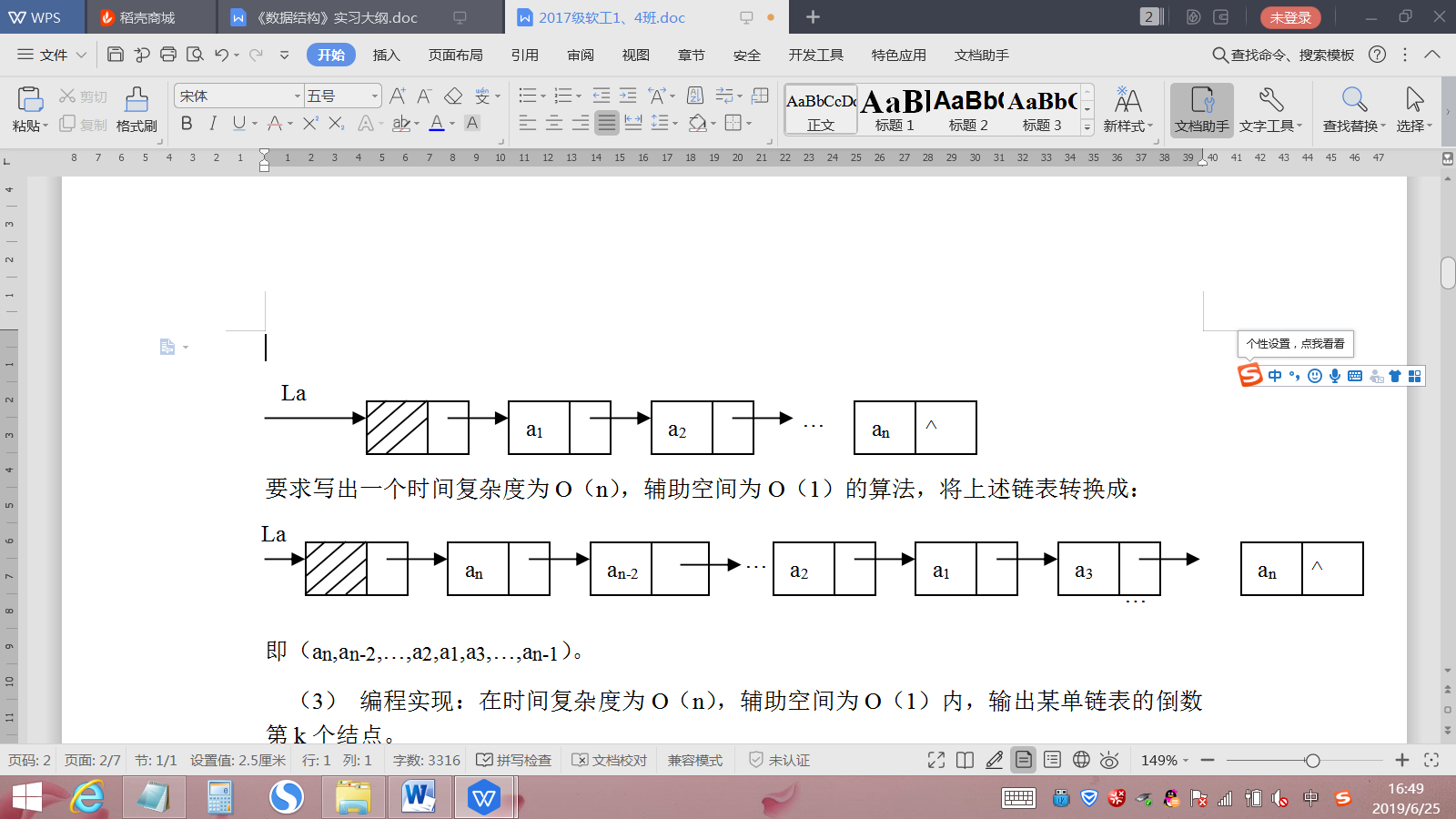
1. 掌握线性表的顺序存储，编写高效算法。
2. 掌握线性表链式存储以及链表基本操作。

**二 实验内容**

* 1. 编写一个程序，其功能是：在一个非递减的顺序表中，删除所有值相等的多余元素，并输出删除前后的元素。要求时间复杂度为O(n)，空间复杂度为O(1)。
  2. 已知有如下单链表（a1,a2,…,an），n为偶数。



要求写出一个时间复杂度为O（n），辅助空间为O（1）的算法，将上述链表转换成：



即（an,an-2,…,a2,a1,a3,…,an-1）。

(3)链表应用：一元多项式运算器。

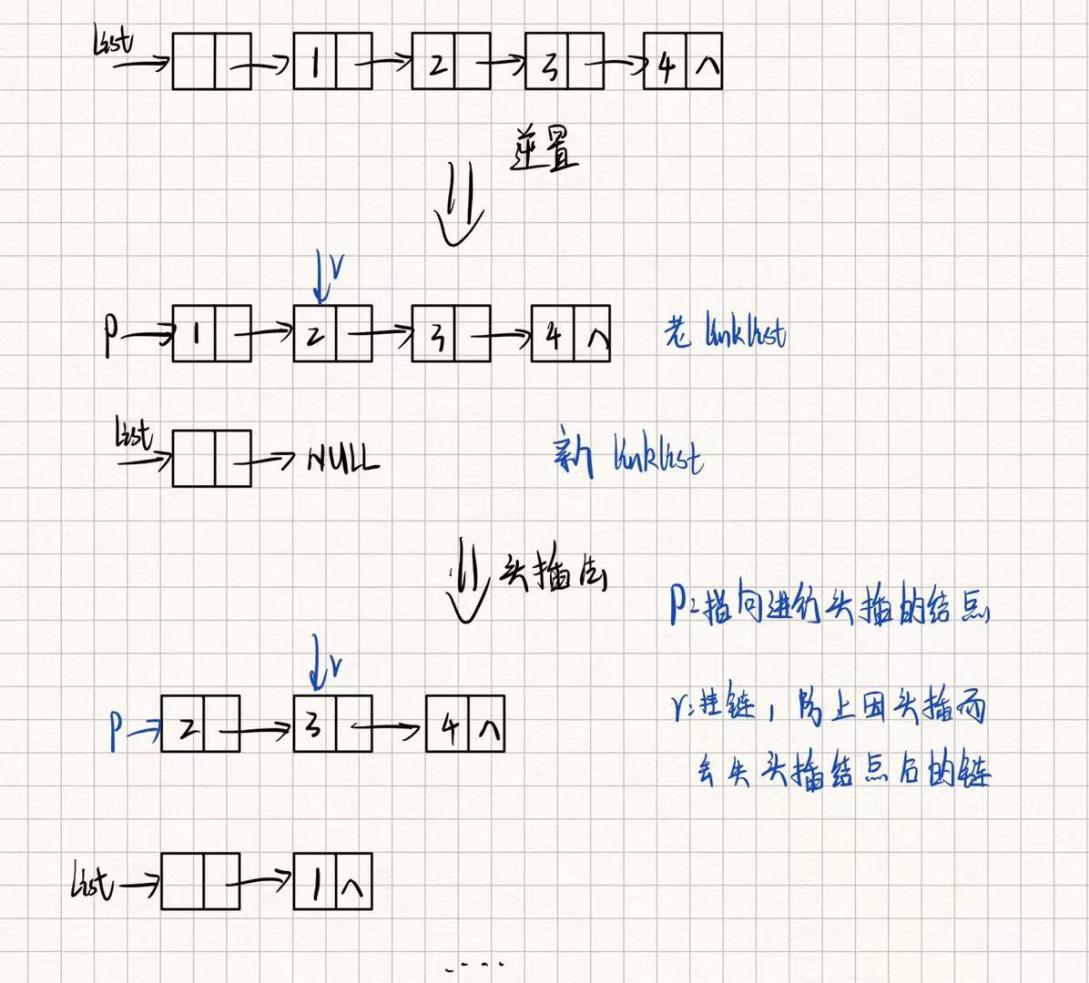
基本要求：

（1） 输入并建立多项式，并用友好的界面显示多项式，如，8x3-6x2+8显示为8x^3-6x^2+8；

（2）计算两个多项式的加法和减法；

（3）给定x，计算多项式在x处的值。

**三 实验设计**

* 问题1
  + 问题分析——顺序表为非递减顺序表，说明相邻元素（前者与后者）不是小于就是相等关系。问题是删除相同元素，并输出删除前后的元素。解决该问题的关键在于如何删除相同元素，并且还必须使时间复杂度为O(n),空间复杂度为O(1)。即不能开辟一片新的内存存放数据。所以采用在原数组的基础上对数组进行覆盖。
  + 算法思路——双循环变量法（双指针法）——定义两个循环变量i、j指向该顺序表。循环变量i用来遍历整个顺序表，另一个循环变量j用来更新顺序表中的数据。i变量顺序遍历数组，若前者与后者不相等，则将前者元素赋给当前j变量所指位置，与此同时i++、j++。若前后相等，则只对i变量进行i++。当i变量遍历结束时，修改线性表的尾指针（最后一个元素的下标）即可。
* 问题2
  + 问题分析——本题所要做的，是将一个链表进行翻转，并且时间复杂度为O（n），辅助空间为O（1）。问题的关键在于如何将链表反转，而且还不能引入新的链表来存储反转后的链表。所以需要在原链表的基础上进行操作。反转链表有两种思路：第一种思路是只交换数值，不改变结点顺序，以达到反转链表的假象。另一种思路是改变链表结点的顺序。交换数值需要采用首尾元素交换的算法思路。本题采用改变结点顺序来实现反转链表。
  + 算法思路——与头插法思路相近（链表就地头插法）。将链表分成两个链表，新链表和老链表。将老链表的结点向新链表进行头插操作(头插操作的性质是链表顺序和输入顺序相反，因此会产生链表翻转效果)。

**四 实验代码**

* 问题一——顺序表删除重复元素
  + 头文件——ArrayList.h

#include**<**stdio**.**h**>**#ifndef ARRAYLIST\_H *//包含守卫*  
#define ARRAYLIST\_H  
  
#define OK 1 *//宏定义*#define ERROR 0  
  
*//顺序表的定义*#define MAXSIZE 100  
  
**typedef** **struct{**    **int** elem**[**MAXSIZE**];**    **int** last**;  
}**SeqList**;***//顺序表相关操作***int** max**(int** a**,** **int** b**);  
int** Locate**(**SeqList L**,** **int** e**);  
int** DelList**(**SeqList **\***L**,** **int** i**,** **int** **\***e**);  
int** InsList**(**SeqList **\***L**,** **int** i**,** **int** e**);  
void** mergeList**(**SeqList **\***LA**,** SeqList **\***LB**,** SeqList **\***LC**);**#endif

* + 源文件
    - main.c

#include**<**stdio**.**h**>**#include**"List.h"***//引入线性表文件（里面包含了顺序表和链表的定义以及基本操作）  
  
//删除重复元素函数***void** DelDuplicateList**(**SeqList **\***num**)** **{**    **int** i **=** 0**,** j **=** 0**;** *//双循环变量*    **for** **(**i **=** 0**;** i **<=** num**->**last**;** i**++)** **{**        **if** **(**num**->**elem**[**i**]** **!=** num**->**elem**[**j**])** **{** *//二者不相等在j所指位置下一位置插入该元素*            num**->**elem**[**j **+** 1**]** **=** num**->**elem**[**i**];**            j**++;**        **}** **else** **{** *//相等则不需要插入元素，移动i，判断下一个元素*            **continue;***//跳过当前循环*        **}**    **}**    num**->**last **=** j**;** *//最终j的值即为最后一个元素的下标*    **return;  
};***//打印函数***void** PrintSeqList**(**SeqList num**)** **{**    **for** **(int** i **=** 0**;** i **<=** num**.**last**;** i**++)** **{**        printf**("%d ",** num**.**elem**[**i**]);**    **}**    printf**("**\n**");**    **return;  
}  
  
  
int** main**()** **{**    SeqList nums **=** **{{**1**,** 2**,** 2**,** 3**,** 4**,** 4**,** 6**,** 6**,** 7**,** 8**},** 9**};** *//结构体赋值（结构体在定义时不可赋值，因为未开辟内存）*    PrintSeqList**(**nums**);**    *//删除重复元素*    DelDuplicateList**(&**nums**);**    PrintSeqList**(**nums**);**    **return** 0**;  
}**

* + - ArrayList\_Function.c——顺序表相关操作

#include**"ArrayList.h"***//线性表顺序存储的基本操作***int** max**(int** a**,** **int** b**)** **{**    **if** **(**a **>=** b**)** **{**        **return** a**;**    **}** **else** **{**        **return** b**;**    **}  
}***//查找***int** Locate**(**SeqList L**,** **int** e**)** **{**    **int** i **=** 0**;**    **while** **((**i **<=** L**.**last**)** **&&** **(**L**.**elem**[**i**]** **!=** e**))** **{**        i**++;**    **}**    **if** **(**i **<=** L**.**last**)** **{**        **return** i **+** 1**;** *//返回查找位置*    **}** **else** **{**        **return** **-**1**;***//未找到*    **}  
}***//删除***int** DelList**(**SeqList **\***L**,** **int** i**,** **int** **\***e**)** **{**    **int** k**;**    **if** **((**i **<** 1**)** **||** **(**i **>** L**->**last **+** 1**))** **{**        printf**("插入位置i不合法**\n**");**        **return** ERROR**;**    **}**    **\***e **=** L**->**elem**[**i **-** 1**];** *//返回删除的元素*    *//元素前移*    **for** **(**k **=** i**;** i **<=** L**->**last**;** k**++)** **{**        L**->**elem**[**k **-** 1**]** **=** L**->**elem**[**k**];**    **}**    L**->**last**--;***//修改尾指针*    **return** OK**;  
}***//插入***int** InsList**(**SeqList **\***L**,** **int** i**,** **int** e**)** **{**    **int** k**;**    **if** **((**i **<** 1**)** **||** **(**i **>** L**->**last **+** 2**))** **{**        printf**("插入位置i不合法**\n**");**        **return** ERROR**;**    **}**    *//元素后移*    **for** **(**k **=** L**->**last**;** k **>=** i **-** 1**;** k**--)** **{**        L**->**elem**[**k **+** 1**]** **=** L**->**elem**[**k**];**    **}**    L**->**elem**[**i **-** 1**]** **=** e**;** *//插入元素*    L**->**last**++;***//修改尾指针*    **return** OK**;  
}***//合并***void** mergeList**(**SeqList **\***LA**,** SeqList **\***LB**,** SeqList **\***LC**)** **{**    **int** i **=** 0**,** j **=** 0**,** k **=** 0**;**    *//两表重合部分先放小的元素*    **while** **(**i **<=** LA**->**last **&&** j **<=** LB**->**last**)** **{**        **if** **(**LA**->**elem**[**i**]** **<=** LB**->**elem**[**j**])** **{**            LC**->**elem**[**k**]** **=** LA**->**elem**[**i**];**            i**++;**k**++;**        **}** **else** **{**            LC**->**elem**[**k**]** **=** LB**->**elem**[**j**];**            j**++;**k**++;**        **}**    **}**    *//LA表有剩余*    **while** **(**i **<=** LA**->**last**)** **{**        LC**->**elem**[**k**]** **=** LA**->**elem**[**i**];**        i**++;**k**++;**    **}**    *//LB表有剩余*    **while** **(**j **>=** LB**->**last**)** **{**        LC**->**elem**[**k**]** **=** LB**->**elem**[**j**];**        j**++;**k**++;**    **}**    LC**->**last **=** max**(**LA**->**last**,** LB**->**last**)** **+** 1**;  
}**

* 问题二——链表反转
  + 头文件——LinkList.h

#include**<**stdio**.**h**>**#include**<**stdlib**.**h**>**#ifndef LINKLIST\_H  
#define LINKLIST\_H  
  
#define OK 1  
#define ERROR 0  
*//链表的定义  
//结点、链表(指向头结点的一个指针)***typedef** **struct** Node**{**    **int** data**;**    **struct** Node **\***next**;  
}**Node**,\***LinkList**;***//链表相关操作***void** InitList**(**LinkList **\***L**);  
void** CreateFromHead**(**LinkList L**);***//头插法***void** CreateFromTail**(**LinkList L**);***//尾插法*Node **\*** Get**(**LinkList L**,int** i**);***//按序号查找*Node **\*** LocateLinkList**(**LinkList L**,int** key**);***//按值查找***int** ListLength**(**LinkList L**);***//求链表长度***int** InsLinkList**(**LinkList L**,int** i**,int** e**);***//插入元素***int** DelLinkList**(**LinkList L**,int** i**,int** **\***e**);***//删除元素*LinkList mergeLinkList**(**LinkList LA**,**LinkList LB**);***//链表合并*#endif

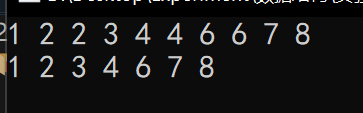
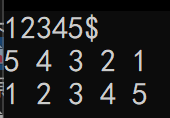
* + 源文件
    - main.c

#include**<**stdio**.**h**>**#include**"LinkList.h"***//链表打印***void** PrintList**(**LinkList list**)** **{**    Node **\***p **=** list**->**next**;***//p指向首结点(头结点的下一个元素)*    **while** **(**p **!=** NULL**)** **{**        printf**("%c ",** p**->**data**);**        p **=** p**->**next**;** *//p后移*    **}**    printf**("**\n**");**    **return;  
}***//链表反转函数  
//算法思路——链表就地逆置，可看成一个老链表的结点向新链表进行头插操作(头插操作的性质是链表顺序和输入顺序相反)***void** ReverseList**(**LinkList list**)** **{**    Node **\***r**,** **\***p**;**    *//老链表*    r **=** list**->**next**;***//r的作用是为了暂时保存下一个进行头插的结点地址,防止链表丢失（挂链）*    p **=** r**;** *//p始终指向需要进行头插的元素*    *//新链表——没有结点，为空*    list**->**next **=** NULL**;**    *//头插法*    **while** **(**p **!=** NULL**)** **{**        r **=** p**->**next**;***//r保存下一个进行头插法的结点地址*        *//将p结点所指向的元素利用头插法插入到list链表*        p**->**next **=** list**->**next**;**        list**->**next **=** p**;**        p **=** r**;** *//指向下一个需要头插的结点*    **}  
  
}  
  
int** main**()** **{**    LinkList list**;**    InitList**(&**list**);**    *//头插法*    CreateFromHead**(**list**);***//输入'$'字符串代表输入结束*    PrintList**(**list**);**    ReverseList**(**list**);***//翻转链表*    PrintList**(**list**);**    **return** 0**;  
}**

* + - LinkList\_Function.c

#include**"LinkList.h"***//建立空链表***void** InitList**(**LinkList **\***L**)** **{**    **\***L **=** **(**LinkList**)**malloc**(sizeof(**Node**));**    **(\***L**)->**next **=** NULL**;** *//空链表***}  
  
void** CreateFromHead**(**LinkList L**)** **{**    **char** c**;**    **int** flag **=** 1**;**    **while** **(**flag**)** **{**        c **=** getchar**();**        **if** **(**c **!=** '$'**)** **{**            *//创建结点并赋值*            Node **\***s **=** **(**Node**\*)**malloc**(sizeof(**Node**));**            s**->**data **=** c**;**            s**->**next **=** L**->**next**;** *//头插法——在头结点后面插入新结点*            L**->**next **=** s**;**        **}** **else** **{**            flag **=** 0**;**        **}**    **}  
}  
  
void** CreateFromTail**(**LinkList L**)** **{**    Node **\***r**;***//r指向当前链表的最后一个元素*    **char** c**;**    **int** flag **=** 1**;**    r **=** L**;**    **while** **(**flag**)** **{**        c **=** getchar**();**        **if** **(**c **!=** '$'**)** **{**            *//创建结点并赋值*            Node **\***s **=** **(**Node**\*)**malloc**(sizeof(**Node**));**            s**->**data **=** c**;**            s**->**next **=** NULL**;**            r**->**next **=** s**;**            r **=** r**->**next**;** *//r后移*        **}** **else** **{**            flag **=** 0**;**            r**->**next **=** NULL**;**        **}**    **}  
}**Node **\*** Get**(**LinkList L**,** **int** i**)** **{**    **int** j**;**    Node **\***p**;**    **if** **(**i **<=** 0**)** **{**        **return** NULL**;**    **}**    p **=** L**;**    j **=** 0**;**    **while** **((**p**->**next **!=** NULL**)** **&&** **(**j **<** i**))** **{**        p **=** p**->**next**;** *//p后移一个结点*        j**++;**    **}**    **if** **(**i **==** j**)** **{**        **return** p**;**    **}** **else** **{**        **return** NULL**;**    **}  
}**Node **\*** LocateLinkList**(**LinkList L**,** **int** key**)** **{**    Node **\***p**;**    p **=** L**->**next**;**    **while** **(**p **!=** NULL**)** **{**        **if** **(**p**->**data **!=** key**)** **{**            p **=** p**->**next**;**        **}** **else** **{**            **break;**        **}**    **}**    **return** p**;  
}  
  
int** ListLength**(**LinkList L**)** **{**    Node **\***p**;**    p **=** L**->**next**;**    **int** j **=** 0**;**    *//当p为最后一个结点所指向的下一个结点地址（即NULL）时，循环结束*    **while** **(**p **!=** NULL**)** **{**        p **=** p**->**next**;**        j**++;**    **}**    **return** j**;  
}  
  
int** InsLinkList**(**LinkList L**,** **int** i**,** **int** e**)** **{**    Node **\***pre **=** L**;***//pre指向删除结点的前一个结点*    **int** k **=** 0**;**    **if** **(**i **<** 0**)** **{**        **return** ERROR**;**    **}**    *//pre最多指向最后一个结点，不能指向最后一个结点所指向的下一个结点的地址（即NULL），因为可能存在在最后一个位置插入元素*    **while** **(**pre **!=** NULL **&&** k **<** i **-** 1**)** **{**        pre **=** pre**->**next**;**        k**++;**    **}**    *//pre不能指向最后一个结点*    **if** **(**pre **==** NULL**)** **{**        printf**("插入位置不合理!");**        **return** ERROR**;**    **}**    Node **\***s **=** **(**Node**\*)**malloc**(sizeof(**Node**));**    s**->**data **=** e**;**    *//插入新结点*    s**->**next **=** pre**->**next**;**    pre**->**next **=** s**;**    **return** OK**;  
}  
  
int** DelLinkList**(**LinkList L**,** **int** i**,** **int** **\***e**)** **{**    Node **\***pre **=** L**,** **\***r**;**    **int** k **=** 0**;**    **while** **(**pre**->**next **!=** NULL **&&** k **<** i **-** 1**)** **{**        pre **=** pre**->**next**;**        k**++;**    **}**    *//循环结束后，pre指向删除元素的前一个结点，k为i-1*    *//pre不能指向最后一个结点的下一个结点（因为最后一个结点可删除）*    **if** **(**pre**->**next **==** NULL**)** **{**        printf**("删除结点的位置i不合理!");**        **return** ERROR**;**    **}**    r **=** pre**->**next**;**    pre**->**next **=** r**->**next**;**    **\***e **=** r**->**data**;**    free**(**r**);**    **return** OK**;  
}**LinkList mergeLinkList**(**LinkList LA**,** LinkList LB**)** **{**    Node **\***pa**,** **\***pb**,** **\***r**;**    LinkList LC**;**    pa **=** LA**->**next**;**    pb **=** LB**->**next**;**    LC **=** LA**;**    LC**->**next **=** NULL**;**    r **=** LC**;***//r指向LC表的最后一个结点*    **while** **(**pa **!=** NULL **&&** pb **!=** NULL**)** **{**        **if** **(**pa**->**data **<=** pb**->**data**)** **{**            r**->**next **=** pa**;**            r **=** pa**;**            pa **=** pa**->**next**;**        **}** **else** **{**            r**->**next **=** pb**;**            r **=** pb**;**            pb **=** pb**->**next**;**        **}**    **}**    **if** **(**pa**)** **{**        r**->**next **=** pa**;**    **}** **else** **{**        r**->**next **=** pb**;**    **}**    free**(**LB**);**    **return** LC**;  
}**

**五 实验效果**

* 实验一——顺序表消除重复元素
  + 
* 实验二——链表翻转
  + 
* 实验三——多项式运算

**六 经验分析**