****

**实 验 报 告**

**实验课程： 数据结构与算法实验**

**实验题目： 线性表实现多项式加法**

**任课老师： 吴之旭老师**

**学生姓名： 殷骢睿**

**学 号： 5509121041 \_**

**专业班级： 2021级人工智能实验班 \_**

**2022 年 10 月 11日**

** 南昌大学实验报告**

学生姓名： 殷骢睿 学号： 5509121041 专业班级： 人工智能211班

实验类型：□ 验证 □ 综合 ■ 设计 □ 创新 实验日期： 2022.10.11 实验成绩：

1. **实验目的**

(1) 掌握线性表的定义；

(2) 掌握线性表的基本操作，如建立、查找、插入和删除等；

1. **实验内容**

1. **分别使用顺序表和链表**实现一元多次多项式的存储和加法，并输出多项式，完成如下功能：

(1) 根据用户的输入，按照X的幂次从小到大排列（常数项幂次为0），存储多项式；

(2) 统计多项式中项的数量和最高次幂；

(3) 实现两个多项式的加法；

(4) 按幂次大小从大到小输出多项式；

**三、实验要求**

(1) 程序要添加适当的注释，程序的书写要采用缩进格式。

(2) 程序要具在一定的**健壮性**，即当输入数据非法时，程序也能适当地做出反应，如插入的项幂次方为负数或超过允许的最大幂次方数等等。

(3) 程序要做到界面友好，在程序运行时用户可以根据相应的提示信息进行操作，比如如何结束一个多项式的输入。

(4) 根据实验报告模板详细书写实验报告,在实验报告中给出链表计算二个一元多项式相加的流程图。

(5) 源程序和实验报告打包。顺序表的源程序保存为SqList.cpp，链表的源程序保存为LinkList.cpp，实验报告命名为：实验报告2.doc。源程序和实验报告压缩为一个文件（如果定义了头文件则一起压缩），按以下方式命名：学号姓名.rar，如0814101王五.rar。

**四、顺序表主要实验步骤及程序分析**

**4.1思路**

**ADT UnaryPoly**{

数据对象：D = {p0, p1, ... , pn| n(-N, p(-R}

数据关系：R = {<p0, p1, ..., pn>| p0是一元n次多项式P的0次项的系数, p1是P的1次项的系数, ..., pn是P的n次项的系数}

基本操作：

create(&P, p0, p1, ..., pn)

操作结构：构造一元n次多项式P

calculate(&ans, P, x)

初始条件：P是一元n次多项式, x是实数

操作结构：确定P的元，计算一元n次多项式的值并返回

add(&P, P1, P2)

初始条件：P1, P2是一元多项式

操作结果：返回P1, P2的和

sub(&P, P1, P2)

初始条件：P1, P2是一元多项式

操作结果：返回P1, P2的和

}**ADT UnaryPoly**;

**4.2主要具体步骤（完整代码见附录）**

(1) 插入后，按照指数大小排序，采用顺序表插入方法和冒泡排序实现。

1. **template**<**class** T>
2. **bool** arrList<T>::setValue(**const** T numberValue, **const** T indexValue) {
3. number[indexValue] += numberValue;
4. **return** **true**;
5. }

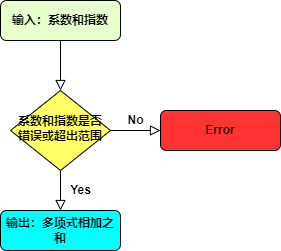
(2) 进行多项式相加时，先逐个判断相同项，当有相同项时，进行相加，若无出现相同项，则放到最后进行处理。

1. Status LocateElem(SqList L, ElemType e,Status (\* compare)(ElemType, ElemType)){
2. //如果找到满足要求的，返回位序，否则返回0
3. **int** i = 1;      //i的初值为第1个元素的位序
4. **int** \*p = L.elem;    //p的初值为第1个元素的存储位置
5. **while**(i<=L.length && !(\* compare)(\*p++, e)) ++i;
6. **if**(i<=L.length) **return** i;
7. **else** **return** 0;
8. }//LocateElem
10. Status PriorElem(SqList L, ElemType cur\_e, ElemType &pre\_e){
11. **int** i = 1;
12. **while**(i<=L.length && cur\_e!=L.elem[i-1]) i++;
13. **if** (i<=L.length){
14. pre\_e = L.elem[i-2];    //当前值的前驱取出来
15. **return** OK;
16. }
17. **else**{
18. **return** ERROR;
19. }
20. }//PriorElem

(3) 若出现异常，程序会自动报错。

1. Status NextElem(SqList L, ElemType cur\_e, ElemType &next\_e){
2. **int** i = 0;
3. **while**(i<L.length && cur\_e!=L.elem[i-1]) i++;
4. **if** (i<L.length){
5. next\_e = L.elem[i]; //当前值的前驱取出来
6. **return** OK;
7. }
8. **else**{
9. **return** ERROR;
10. }
11. }//NextElem

**4.3 流程图**

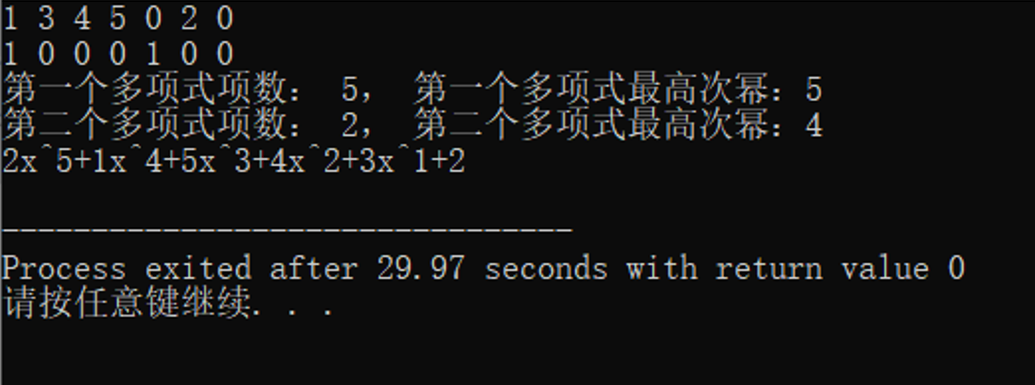
****

**图1 线性表实现多项式相加流程图**

**4.4 程序结果**

输入：两行的多项式的系数

输出：相加多项式，各多项式的项式和最高次幂。



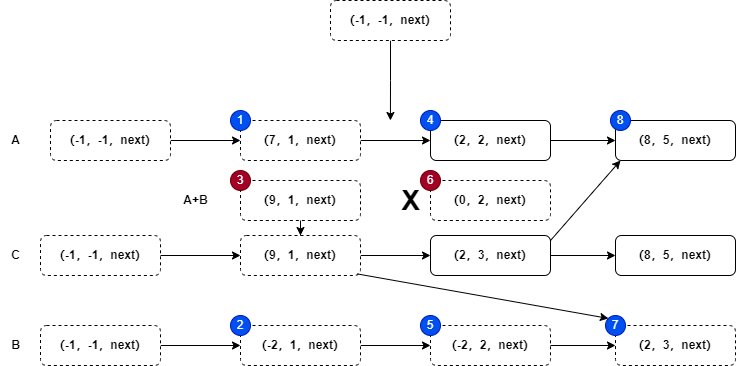
**图2 线性表实现多项式相加程序结果图**

**五、链表主要实验步骤及程序分析**

**5.1思路**

**工作指针 pre、p、qre、q 初始化  
while（p 存在且 q 存在）执行下列三种情况之一：  
　若 p->exp < q->exp：指针 p 后移；  
　若 p->exp > q->exp，则  
　　将结点 q 插到结点 p 之前  
　　指针 p 指向他原指结点的下一个结点；  
　若 p->exp ==q->exp，则  
　　p->coef = p->coef + q->coef  
　　若 p->coef == 0,则执行下列操作，否则指针 p 后移，  
　　　　删除结点 p  
　　　　使指针 p 指向它原指结点的下一个结点  
　　删除结点 q  
　　使指针 q 指向它原指结点的下一个结点  
如果 q 不为空，将结点 q 链接在第一个单链表的后面**。

**5.2流程图**



**图3 链表实现多项式相加流程图**

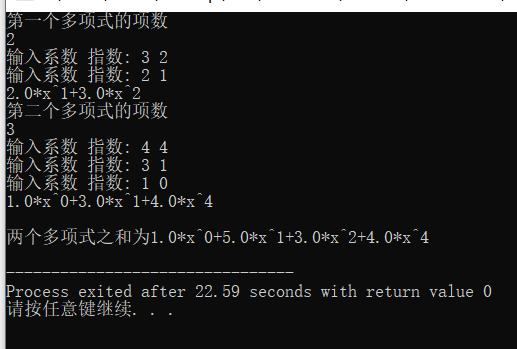
**5.3主要具体步骤（完整代码见附录）**

1. //将L线性链表清空，释放链表空间，保留头结点
2. Status ClearList(LinkList &L){
3. Link p;
4. p = L.tail;
5. **while**(p!=L.head){
6. p=PriorPos(L,p);
7. FreeNode(p->next);
8. }
9. L.len=0;
10. **return** OK;
11. }//ClearList
13. Status InsFirst(LinkList &L, Link s){
14. s->next=L.head->next;
15. **if**(!L.head->next) L.tail=s;
16. L.head->next=s;
17. L.len++;
18. **return** OK;
19. }//InsFirst
21. Status DelFirst(LinkList &L, Link &q){
22. **if**(!L.head->next){
23. **return** ERROR;
24. }
25. q = L.head->next;
26. L.head->next=L.head->next->next;
27. **if**(!L.head->next){
28. L.tail=L.head;
29. }
30. L.len--;
31. **return** OK;
32. }//DelFirst
34. Status Append(LinkList &L,Link s){
35. Link q;
36. q=L.head;
37. **if** (!L.tail){
38. L.head->next=s;
39. //更新尾结点
40. **while**(q->next) q=q->next;
41. L.tail=q;
42. }
43. L.tail->next=s;
44. **while**(q->next) q=q->next;
45. L.tail=q;
46. L.len+=Length(s);
47. **return** OK;
48. }//Append

**5.4 程序结果**

输入：每个多项式项式以及多项式的指数和系数

输出：两个多项式相加之和



**图4 链表实现多项式相加程序设计图**

**六、编写代码时遇到的问题及实验心得体会**

**遇到的问题：**

(1) 链表的掌握并不熟练，提醒了自己平时在学数据结构的过程中要多写代码；

(2) Visual Studio IDE使用不熟练；

(3) 在人机交互上没能想到很好的方案；

**实验心得体会：**

(1) 编程应该联系理论和实际，这次编程中，我发现链表的效率比线性表高出许多，在内存(274KB vs 746KB)和编译时间上(2.87s vs 4.86s)均占有一定的优势，让我感受到了数据结构的魅力。

(2) 老师的题目很难，也让我知道了自己的不足，这次花了很多的时间来完成代码和报告，期间报错约有100余次，为我的Coding能力敲响了警钟。

(3) 这次编程坚决没有上网搜索现成的方案，锻炼了自己的意志力和克服困难的能力，很感谢老师的这次考验。

**七、实验源码**

**附件一：Sqlist.cpp**

1. #include <iostream>
2. #include "status.h"
3. #include "sqlist.h"
5. #define LIST\_INIT\_SIZE 100 //初始分配
6. #define LISTINCREMENT 10    //动态分配增加量
7. **using** **namespace** std;
9. Status InitList\_Sq(SqList &L){
10. //申请内存空间
11. L.elem = (ElemType \*)malloc(LIST\_INIT\_SIZE\***sizeof**(ElemType));
12. **if**(!L.elem) exit(OVERFLOW);     //申请失败，报错
13. L.length = 0;                   //空表长度为0
14. L.listsize = LIST\_INIT\_SIZE;    //初始存储量
15. **return** OK;
16. }//InitList\_Sq
18. Status DestroyList\_Sq(SqList &L){
19. **if**(L.elem) free(L.elem);    //如果线性表存在，则释放空间
20. **if**(!L.elem) exit(OVERFLOW); //如果线性表不存在，则报错
21. **return** OK;
22. }//DestroyList\_Sq
24. Status ClearList\_Sq(SqList &L){
25. L.length = 0;
26. **return** OK;
27. }//ClearList\_Sq
29. **int** ListLength\_Sq(SqList L){
30. **return** L.length;
31. }//ListLength\_Sq
33. Status GetElem\_Sq(SqList L, **int** i, ElemType &e){
34. **if** (i<1 || i>L.length+1) **return** ERROR; //如果i不在当前L中
35. ElemType \*q = &(L.elem[i-1]); //取出位置
36. e = \*q;             //取出元素
37. **return** OK;
38. }//GetElem\_Sq
40. Status LocateElem(SqList L, ElemType e,Status (\* compare)(ElemType, ElemType)){
41. //如果找到满足要求的，返回位序，否则返回0
42. **int** i = 1;      //i的初值为第1个元素的位序
43. **int** \*p = L.elem;    //p的初值为第1个元素的存储位置
44. **while**(i<=L.length && !(\* compare)(\*p++, e)) ++i;
45. **if**(i<=L.length) **return** i;
46. **else** **return** 0;
47. }//LocateElem
49. Status PriorElem(SqList L, ElemType cur\_e, ElemType &pre\_e){
50. **int** i = 1;
51. **while**(i<=L.length && cur\_e!=L.elem[i-1]) i++;
52. **if** (i<=L.length){
53. pre\_e = L.elem[i-2];    //当前值的前驱取出来
54. **return** OK;
55. }
56. **else**{
57. **return** ERROR;
58. }
59. }//PriorElem
61. Status NextElem(SqList L, ElemType cur\_e, ElemType &next\_e){
62. **int** i = 0;
63. **while**(i<L.length && cur\_e!=L.elem[i-1]) i++;
64. **if** (i<L.length){
65. next\_e = L.elem[i]; //当前值的前驱取出来
66. **return** OK;
67. }
68. **else**{
69. **return** ERROR;
70. }
71. }//NextElem
73. Status ListInsert\_Sq(SqList &L, **int** i, ElemType e){
74. **if**(i<1 || i>L.length+1) **return** ERROR;
76. **if**(L.length >= L.listsize){      //当前长度大于等于最大存储容量
77. ElemType \*newbase = (ElemType \*)realloc(L.elem,(L.listsize+LISTINCREMENT)\* **sizeof**(ElemType));
78. **if**(!newbase) exit(OVERFLOW); //分配失败
79. L.elem = newbase;   //新基址
80. L.listsize += LISTINCREMENT;    //记录当前的最大容量
81. }
83. ElemType \*q = &(L.elem[i-1]); //q插入位置
84. **for**(ElemType \*p = &(L.elem[L.length-1]); p >= q; --p) \*(p+1) = \*p; //插入位置及之后的元素右移
86. \*q = e; //插入e
87. ++L.length; //当前长度+1
88. **return** OK;
89. }//ListInsert\_Sq
91. Status ListDelete\_Sq(SqList &L, **int** i, ElemType &e){
92. //线性表中删除第i个元素，并用e返回
93. **if**((i<1)||(i>L.length)) **return** ERROR;
94. ElemType \*p = &(L.elem[i-1]);
95. e = \*p;
96. ElemType \*q = L.elem + L.length -1 ;
97. **for**(++p; p<=q; ++p) \*(p-1)= \*p;  //删除元素之后的元素左移
98. --L.length;
99. **return** OK;
100. }//ListDelete\_Sq

**附件二：Linklist.cpp**

1. #include <iostream>
2. #include "status.h"
3. #include "linklist.h"
5. **using** **namespace** std;
6. /\*---length---求链的长度\*/
7. **int** Length(Link s)
8. {
9. **int** i=0;
10. Link p;
11. p=s;
12. **while**(p->next){
13. p = p->next;
14. i++;
15. }
16. **return** i;
17. }//Length
18. Status compare(**int** a,**int** b)
19. {
20. **return** a-b;
21. }//Compare
23. Status MakeNode(Link &p, ElemType e ){
24. //一个结点申请内存空间
25. p = (Link)malloc(**sizeof**(LNode));
26. **if**(!p) exit(OVERFLOW);  //申请失败，报错
27. p->data=e;
28. p->next=NULL;            //将
29. **return** OK;
30. }//MakeNode
32. **void** FreeNode(Link &p){
33. **if**(!p) exit(OVERFLOW);
34. free(p);
35. }//FreeNode
37. Status InitList(LinkList &L){
38. Link h;
39. ElemType e;
40. MakeNode(h,e);
41. L.head=h;
42. L.head->next=NULL;
43. L.tail=L.head;
44. L.len=0;    //注意不包括头结点
45. **if** (!L.head) exit(OVERFLOW);
46. **return** OK;
47. }//InitList
49. Status DestroyList(LinkList &L){
50. Link p;
51. p = L.tail;
52. **while**(p!=L.head){
53. p=PriorPos(L,p);
54. FreeNode(p->next);
55. }
56. FreeNode(L.head);
57. **return** OK;
58. }//DestroyList
60. //将L线性链表清空，释放链表空间，保留头结点
61. Status ClearList(LinkList &L){
62. Link p;
63. p = L.tail;
64. **while**(p!=L.head){
65. p=PriorPos(L,p);
66. FreeNode(p->next);
67. }
68. L.len=0;
69. **return** OK;
70. }//ClearList
72. Status InsFirst(LinkList &L, Link s){
73. s->next=L.head->next;
74. **if**(!L.head->next) L.tail=s;
75. L.head->next=s;
76. L.len++;
77. **return** OK;
78. }//InsFirst
80. Status DelFirst(LinkList &L, Link &q){
81. **if**(!L.head->next){
82. **return** ERROR;
83. }
84. q = L.head->next;
85. L.head->next=L.head->next->next;
86. **if**(!L.head->next){
87. L.tail=L.head;
88. }
89. L.len--;
90. **return** OK;
91. }//DelFirst
93. Status Append(LinkList &L,Link s){
94. Link q;
95. q=L.head;
96. **if** (!L.tail){
97. L.head->next=s;
98. //更新尾结点
99. **while**(q->next) q=q->next;
100. L.tail=q;
101. }
102. L.tail->next=s;
103. **while**(q->next) q=q->next;
104. L.tail=q;
105. L.len+=Length(s);
106. **return** OK;
107. }//Append
108. Status Remove(LinkList &L,Link &q){
109. **if**(!L.tail) **return** ERROR;
110. q=L.tail;
111. L.tail=PriorPos(L,q);
112. L.tail->next=NULL;
113. **return** OK;
114. }//Remove
115. Status InsBefore(LinkList &L, Link &p,Link s){
116. Link q;
117. q=PriorPos(L,p);
118. s->next=p;
119. q->next=s;
120. **return** OK;
121. }//InsBefore
122. Status InsAfter(LinkList &L, Link &p,Link s){
123. s->next=p->next;
124. p->next=s;
125. **return** OK;
126. }//InsAfter
128. Status SetCurElem(Link &p,ElemType e){
129. p->data=e;
130. **return** OK;
131. }//SetCurElem
132. ElemType GetCurElem(Link p){
133. **return** p->data;
134. }
135. Status ListEmpty(LinkList L){
136. **if**(L.head==L.tail) **return** TRUE;
137. **return** FALSE;
138. }//ListEmpty
139. **int** ListLength(LinkList L){
140. **return** L.len;
141. }//ListLength
142. Position GetHead(LinkList L){
143. **return** L.head;
144. }
145. Position GetLast(LinkList L){
146. **return** L.tail;
147. }
148. Position PriorPos(LinkList L, Link p){
149. Link q;
150. q=L.head;
151. **if**(q->next==p) **return** ERROR;
152. **while**(q->next!=p) q=q->next;
153. **return** q;
154. }//PriorPos
155. Link NextPos(LinkList L, Link p){
156. Link q;
157. **if**(!p->next) {
158. **return** NULL;
159. }
160. q=p->next;
161. **return** q;
162. }
163. //这里的i从0开始
164. Status LocatePos(LinkList L, **int** i, Link &p){
165. p=L.head;
166. **if** (i<=0||i>ListLength(L)) **return** ERROR;
167. **while**(i){
168. p=p->next;
169. i--;
170. }
171. **return** OK;
172. }//LocatePos
174. Position LocateElem(LinkList L, ElemType e, Status (\* compare)(ElemType, ElemType)){
175. **int** i=0;
176. Link p;
177. p=L.head->next;
178. **while**(compare(p->data,e)&&p){
179. p=p->next;
180. ++i;
181. }
182. **if**(!p) **return** ERROR;    //找不到指定元素
183. **return** p;
184. }//LocateElem
185. //void ListTraverse(LinkList L,Status(\* visit)())
186. //{
187. //    Link p;
188. //    p=L.head;
189. //    while(!visit(p))
190. //        p=p->next;
191. //}//ListTraverse