《数据结构与算法》实验报告

实验名称	一元多项式的加法和乘法				
姓名	叶鹏	学号	20020007095	日期	2022/3/10
实验内容	1、通过键盘随机输入两个多项式 P（x）和 Q（x）的内容。 2、输出结果要有 P（x)、Q（x）以及他们的和。 3、输入输出多项式的格式可自行定义。				
实验目的	实现一元多项式的加法和乘法				

实验步骤

1. 一元多项式的加法
2. 考虑到使用链表结构实现一元多项式，格式为 coef x^expn X 作为未知量保留，只需用户输入 coef 系数，与 expn 指数即可，建立链表

```
5   struct LinkList
6   {
7       float coef;
8       int expn;
9       LinkList* next;
10      LinkList(float coef, int expn, LinkList* next) {
11          this->coef = coef;
12          this->expn = expn;
13          this->next = next;
14      }
15  };
```

picture 1 **节点**

```
17  class polynomial
18  {
19  public:
20      polynomial();
21      ~polynomial();
22
23      LinkList* getDump() { return dumpNode; }
24      LinkList* getHead();
25      void setHead(LinkList* node) { dumpNode->next = node; }
26      LinkList* getTail();
27      int getLength() { return getTail()->expn; }
28
29  private:
30      LinkList* dumpNode;
31  };
```

picture 2 **链表**

3. 对于多项式的加法，我们考虑 Pa + Pb，将加和结果整理到 Pa 的空间，接着删除 Pb，于是 Pa 的结果便是加法的和。
4. 加法的运算逻辑为，相同系数的项相加，如果 Pb 中存在 Pa 中未有的项，便按照大小顺序插入到 Pa 中，如果 Pa 中的某一项与 Pb 中的某一项系数加和结果为 0 ，则在 Pa 中删除该项，依照这个思路，完成代码的编写

```
205     //完成多项式相加运算,即Pa=Pa+Pb,并销毁一元多项式Pb
206     void AddPolyn(polynomial& Pa, polynomial& Pb) {
207
208         LinkList* nodeA = Pa.getHead();
209         LinkList* nodeB = Pb.getHead();
210
211         cout << "多项式: ";
212         PrintPolyn(Pa);
213         cout << "多项式: ";
214         PrintPolyn(Pb);
215
216         while (nodeA && nodeB) {
217             if (nodeA->expn < nodeB->expn) {
218                 nodeA = nodeA->next;
219             }
220             else if (nodeA->expn == nodeB->expn) {
221                 float coef = nodeA->coef + nodeB->coef;
222                 if (coef == 0.0) {
223                     LinkList* delNode = nodeA;
224                     nodeA = nodeA->next;
225                     deleteNode(Pa, delNode);
226                 }
227                 else {
228                     nodeA->coef = coef;
229                     nodeA = nodeA->next;
230                 }
231                 LinkList* delNode = nodeB;
232                 nodeB = nodeB->next;
233                 deleteNode(Pb, delNode);
234
235             }
236             else if (nodeA->expn > nodeB->expn) {
237                 insertNode(Pa, nodeB);
238                 LinkList* delNode = nodeB;
239                 nodeB = nodeB->next;
240                 deleteNode(Pb, delNode);
241             }
242         }
243
244         if (!ListEmpty(Pb))
245             Append(Pa, Pb);
246
247         cout << "加和结果: ";
248         PrintPolyn(Pa);
249
250         DestoryPolyn(Pb);
251
252         return;
253     }
```

picture 3 **多项式加法算法**

5. 一元多项式的乘法
6. 与加法类似，乘法中所用到的诸多函数在加法中已经实现
7. 与加法不同的是，乘法需要将 Pa 中的每一项与 Pb 中的每一项进行乘法运算，即系数相乘，指数相加，在此过程中会产生几个边界条件
8. 系数为 0，输出时忽略此项
9. 系数为 1，输出时忽略系数
10. 指数为 0，输出时忽略指数
11. 指数为 1，输出时忽略指数上标

为这些条件添加特殊判定，将相乘结果逐个相加生成一个新的多项式 Pc，将 Pc 作为函数结果返回

```
260  polynomial* MutiPolyn(polynomial& Pa, polynomial& Pb) {
261      polynomial* ans = new polynomial();
262
263      LinkList* cur = Pa.getHead();
264
265      while (cur) {
266          LinkList* target = Pb.getHead();
267          while(target){
268              LinkList* newNode = new LinkList(cur->coef * target->coef,
269                  cur->expn + target->expn, nullptr);
270              LinkList* point = LocateElem(*ans, newNode);
271              if (point) {
272                  if (point->coef + newNode->coef == 0) {
273                      deleteNode(*ans, point);
274                  }
275                  else {
276                      point->coef += newNode->coef;
277                  }
278              }
279              else {
280                  insertNode(*ans, newNode);
281              }
282              target = target->next;
283          }
284
285          cur = cur->next;
286      }
287
288      return ans;
289  }
290
```

picture 4 **多项式乘法**

12. 运行结果
13. 多项式加法
14. (x^2 + 2x^3) + (3x^4 + 4x^5 + 5x^6)

Microsoft Visual Studio 调试控制台

```
1 2
2 3
3 4
4 5
5 6
多项式：P(x) = x^2 + 2x^3
多项式：P(x) = 3x^4 + 4x^5 + 5x^6
加和结果：P(x) = x^2 + 2x^3 + 3x^4 + 4x^5 + 5x^6

C:\Users\ASUS\Documents\数据结构\实验\第二、三章\一
0。
```

15. (x^2 + 2x^3) + (-x^2 - 2x^3 + 3x^4)

Microsoft Visual Studio 调试控制台

```
1 2
2 3
-1 2
-2 3
3 4
多项式：P(x) = x^2 + 2x^3
多项式：P(x) = -x^2 - 2x^3 + 3x^4
加和结果：P(x) = x^4

C:\Users\ASUS\Documents\数据结构\实
```

16. (x^2 + 2x^3) + (x^2 + 2x^3 + 3x^4)

Microsoft Visual Studio 调试控制台

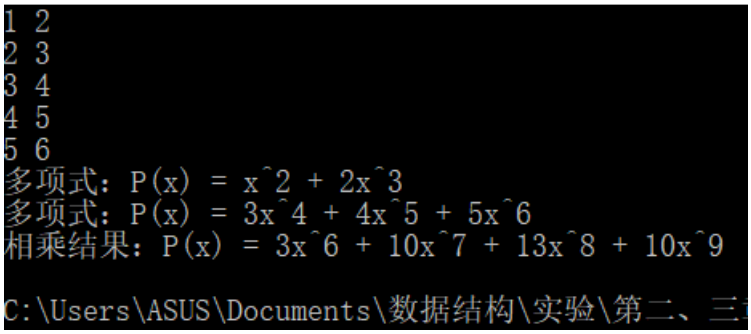
```
1 2
2 3
1 2
2 3
3 4
多项式：P(x) = x^2 + 2x^3
多项式：P(x) = x^2 + 2x^3 + 3x^4
加和结果：P(x) = 2x^2 + 4x^3 + 3x^4

C:\Users\ASUS\Documents\数据结构\实
```

17. 多项式乘法

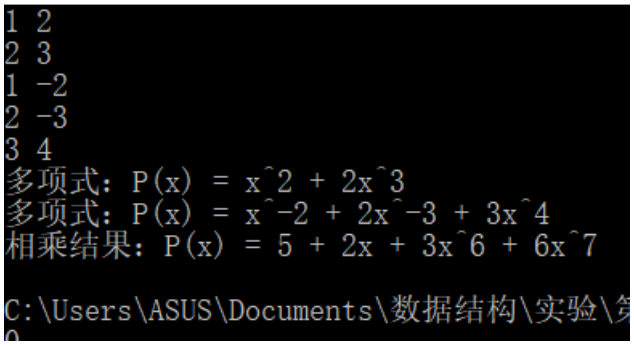
18. (x^2 + 2x^3) * (3x^4 + 4x^5 + 5x^6)

Microsoft Visual Studio 调试控制台



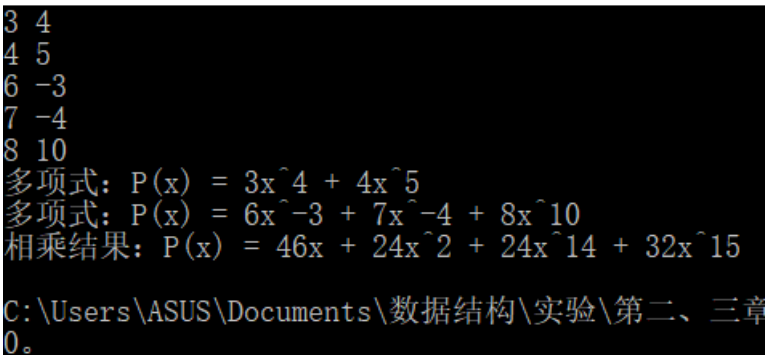
19. (x^2 + 2x^3) * (x^-2 + 2x^-3 + 3x^4)

Microsoft Visual Studio 调试控制台



20. (3x^4 + 4x^5) * (6x^-3 + 7x^-4 + 8x^10)

Microsoft Visual Studio 调试控制台



21. 源代码

```
#pragma once
#include <iostream>
using namespace std;

struct LinkList
{
    float coef;
    int expn;
    LinkList* next;
    LinkList(float coef, int expn, LinkList* next) {
```

```
32.         this->coef = coef;
33.         this->expn = expn;
34.         this->next = next;
35.     }
36. };
37.
38. class polynomial
39. {
40. public:
41.     polynomial();
42.     ~polynomial();
43.
44.     LinkList* getDump() { return dumpNode; }
45.     LinkList* getHead();
46.     void setHead(LinkList* node) { dumpNode->next = node; }
47.     LinkList* getTail();
48.     int getLength() { return getTail()->expn; }
49.
50. private:
51.     LinkList* dumpNode;
52. };
53.
54. polynomial::polynomial()
55. {
56.     dumpNode = new LinkList(0, 0, nullptr);
57. }
58.
59. polynomial::~polynomial()
60. {
61. }
62.
63. inline LinkList* polynomial::getHead()
64. {
65.     if (dumpNode->next == nullptr) {
66.         return nullptr;
67.     }
68.     return dumpNode->next;
69. }
70.
71. inline LinkList* polynomial::getTail()
72. {
73.     LinkList* curNode = getDump();
74.     while (curNode->next != nullptr) {
75.         curNode = curNode->next;
```

```
    }
    return curNode;
}


LinkList* GetHead(polynomial& p) {
    return p.getHead();
}

LinkList* LocateElem(polynomial& p, LinkList* node) {
    LinkList* cur = p.getHead();
    while (cur != nullptr) {
        if (node->expn == cur->expn) {
            return cur;
        }
        cur = cur->next;
    }
    return nullptr;
}

//输入 m 项的系数和指数，建立表示一元多项式的有序链表 P
void CreatPolyn(polynomial& p, int m) {
    float coef;
    int expn;
    while (m)
    {
        cin >> coef >> expn;
        LinkList* newNode = new LinkList(coef, expn, nullptr);
        LinkList* point = LocateElem(p, newNode);
        if (!point) {
            if (p.getHead() == nullptr)
                p.setHead(newNode);
            else
                p.getTail()->next = newNode;
        }
        m--;
    }
}

//返回一元多项式 P 中的项数
int PolynLenght(polynomial p) {
    return p.getLength();
}
```

```
120.
121. void DestoryHelper(LinkList* curNode) {
122.     if (curNode->next == nullptr) {
123.         delete curNode;
124.         return;
125.     }
126.     DestoryHelper(curNode->next);
127.     delete curNode;
128.     return;
129. }
130.
131. //销毁一元多项式 P
132. void DestoryPolyn(polynomial& p) {
133.     LinkList* curNode = p.getHead();
134.     if (curNode == nullptr)
135.         return;
136.     DestoryHelper(curNode);
137.     return;
138. }
139.
140. //打印输出一元多项式 P
141. void PrintPolyn(polynomial p) {
142.     cout << "P" /*<< p.getLength()*/ << "(x) = ";
143.     if (p.getHead() == nullptr) {
144.         cout << "多项式不存在！请先创建！" << endl;
145.         return;
146.     }
147.     LinkList* curNode = p.getHead();
148.     while (curNode != nullptr) {
149.         if (curNode != p.getHead()) {
150.             if (curNode->coef > 0)
151.                 cout << " + ";
152.             else
153.                 cout << " - ";
154.         }
155.
156.         if (fabs(curNode->coef) != 1 || curNode == p.getHead()) {
157.             if (curNode == p.getHead()) {
158.                 if (curNode->coef == -1) cout << "-";
159.                 else if (curNode->coef != 1) cout << curNode->coef;
160.             }
161.             else
162.                 cout << fabs(curNode->coef);
163.         }
```

```
164.
165.         if (curNode->expn != 0) {
166.             cout << "x";
167.             if (curNode->expn != 1) {
168.                 cout << "^" << curNode->expn;
169.             }
170.         }
171.
172.         curNode = curNode->next;
173.     }
174.     cout << endl;
175. }
176.
177. void insertNode(polynomial& p, LinkList* node) {
178.     if (p.getDump()->next == nullptr) {
179.         p.getDump()->next = node;
180.         return;
181.     }
182.
183.     LinkList* curNode = p.getDump();
184.     while (curNode != nullptr)
185.     {
186.         if (curNode->next == nullptr && node->expn > curNode->expn) {
187.             curNode->next = node;
188.             return;
189.         }
190.         if (node->expn > curNode->expn && node->expn < curNode->next->exp
    n) {
191.             LinkList* newNode = new LinkList(node->coef, node->expn, null
    ptr);
192.             newNode->next = curNode->next;
193.             curNode->next = newNode;
194.             return;
195.         }
196.         curNode = curNode->next;
197.     }
198. }
199.
200. bool ListEmpty(polynomial p) {
201.     return p.getHead() == nullptr;
202. }
203.
204. void Append(polynomial& pa, polynomial& pb) {
205.     LinkList* curNode = pa.getDump();
```

```
206.     while (curNode->next != nullptr) {
207.         curNode = curNode->next;
208.     }
209.     LinkList* bNode = pb.getHead();
210.     while (bNode != nullptr) {
211.         LinkList* newNode = new LinkList(bNode->coef, bNode->expn, nullpt
    r);
212.         curNode->next = newNode;
213.         curNode = curNode->next;
214.         bNode = bNode->next;
215.     }
216. }
217.
218. //删除节点
219. void deleteNode(polynomial& p, LinkList* node) {
220.     LinkList* curNode = p.getDump();
221.     while (curNode != nullptr) {
222.         if (curNode->next == node) {
223.             curNode->next = node->next;
224.             delete(node);
225.             return;
226.         }
227.         curNode = curNode->next;
228.     }
229.     cout << "未找到节点！" << endl;
230.     return;
231. }
232.
233. //完成多项式相加运算,即 Pa=Pa+Pb,并销毁一元多项式 Pb
234. void AddPolyn(polynomial& Pa, polynomial& Pb) {
235.
236.     LinkList* nodeA = Pa.getHead();
237.     LinkList* nodeB = Pb.getHead();
238.
239.     cout << "多项式：";
240.     PrintPolyn(Pa);
241.     cout << "多项式：";
242.     PrintPolyn(Pb);
243.
244.     while (nodeA && nodeB) {
245.         if (nodeA->expn < nodeB->expn) {
246.             nodeA = nodeA->next;
247.         }
248.         else if (nodeA->expn == nodeB->expn) {
```

```
            float coef = nodeA->coef + nodeB->coef;
            if (coef == 0.0) {
                LinkList* delNode = nodeA;
                nodeA = nodeA->next;
                deleteNode(Pa, delNode);
            }
            else {
                nodeA->coef = coef;
                nodeA = nodeA->next;
            }
            LinkList* delNode = nodeB;
            nodeB = nodeB->next;
            deleteNode(Pb, delNode);

        }
        else if (nodeA->expn > nodeB->expn) {
            insertNode(Pa, nodeB);
            LinkList* delNode = nodeB;
            nodeB = nodeB->next;
            deleteNode(Pb, delNode);
        }
    }

    if (!ListEmpty(Pb))
        Append(Pa, Pb);

    cout << "加和结果: ";
    PrintPolyn(Pa);

    DestoryPolyn(Pb);

    return;
}

polynomial* MutiPolyn(polynomial& Pa, polynomial& Pb) {
    polynomial* ans = new polynomial();

    LinkList* cur = Pa.getHead();

    while (cur) {
        LinkList* target = Pb.getHead();
        while(target){
            LinkList* newNode = new LinkList(cur->coef * target->coef,
                cur->expn + target->expn, nullptr);
```

```
293.             LinkList* point = LocateElem(*ans, newNode);
294.             if (point) {
295.                 if (point->coef + newNode->coef == 0) {
296.                     deleteNode(*ans, point);
297.                 }
298.                 else {
299.                     point->coef += newNode->coef;
300.                 }
301.             }
302.             else {
303.                 insertNode(*ans, newNode);
304.             }
305.             target = target->next;
306.         }
307.
308.         cur = cur->next;
309.     }
310.
311.     return ans;
312. }
```

实验总结	本次实验通过使用链表模拟多项式的加法与乘法，熟悉了链表这一数据结构，对链表的简单操作加以掌握，并尝试使用此数据结构来解决实际问题，使我们了解到链表的强大之处