

《数据结构》上机报告

2020 年 10 月 13 日

姓名： 林日中 学号： 1951112 班级： 10072602 得分：

实验题目	链表实验报告	
问题描述	<p>链表是一种物理存储单元上非连续、非顺序的存储结构，数据元素的逻辑顺序是通过链表中的指针链接次序实现的。链表由一系列结点（链表中每一个元素称为结点）组成，结点可以在运行时动态生成。每个结点包括两个部分：一个是存储数据元素的数据域，另一个是存储下一个结点地址的指针域。相比于线性表顺序结构，操作复杂。由于不必须按顺序存储，链表在插入的时候可以达到 $O(1)$ 的复杂度，比另一种线性表顺序表快得多，但是查找一个节点或者访问特定编号的节点则需要 $O(n)$ 的时间，而线性表和顺序表相应的时间复杂度分别是 $O(\log n)$ 和 $O(1)$。</p> <p>实验目的：</p> <ol style="list-style-type: none"> 1、掌握线性表的链式表示（单链表、循环链表、双向循环链表）； 2、掌握链表实现线性表的基本操作，如建立、查找、插入、删除、去重、逆置、头部插入、尾部插入、销毁等； 3、掌握有序线性表的插入、删除、合并操作，及一元多项式的表示、相加和乘法等； 	
基本要求	<ol style="list-style-type: none"> (1) 链表的基本操作； (2) 链表的逆置； (3) 链表的去重； (4) 一元多项式的表示、相加和乘法； 	
	已完成基本内容（序号）：	1, 2, 3, 4
选做要求		
	已完成选做内容（序号）：	无
数据结构设计	<p>本次上机分别针对实验内容 1, 2, 3 和实验内容 4 分别设计了两组数据结构。</p> <p>第一组数据结构包括基础数据结构结点 <code>struct Node</code>，分别存储了表示存储数据的 <code>ElemType data</code> 和下一结点指针的 <code>Node *next</code>；和由表示链表的 <code>struct LinkList</code>，存储了头指针 <code>pNode head</code>。</p> <p>第二组数据结构包括多项式的项 <code>struct term</code>，存储了系数 <code>double coef</code> 和指数 <code>int expn</code>；基础数据结构结点 <code>struct Node</code>，存储了当前结点的数据 <code>ElemType data</code> 和存储下一结点指针的 <code>Node *next</code>；和由表示链表的 <code>struct polynominal</code>，存储了头指针 <code>pNode head</code>。</p>	

功能
(函数)
说明

第一组:

```
typedef struct Node
{
    ElemType data = 0;
    struct Node *next = nullptr;
    Node() { cin >> data; }
    Node(const ElemType &e) : data(e) {}
} * pNode;

struct LinkList
{
    pNode head = nullptr;
    LinkList(const size_t n) : head(new Node(0))...
    LinkList(const size_t n, int) : head(new Node(0))...
    ~LinkList() {}
    LinkList &clear()...
    Status insert(const size_t n, const ElemType &e)...
    Status _delete(const size_t n)...
    Status searchByValue(const ElemType &e)...
    size_t count()...
    size_t display()...
    Status distinguishing()...
    Status reverse(const size_t O, const size_t D)...
}; // struct LinkList
```

`struct Node` 的两个成员函数都为构造函数，并且是同名的重载函数，功能是直接通过由实参传入、或从输入流对象 `cin` 中读入的方式给结点的 `data` 赋初值。

`struct LinkList` 的两个构造函数互为重载，分别实现了头插法和尾插法，通过第一个参数后有没有一个整数作为区分。函数 `clear()` 实现了对链表中所有元素的清空，使其恢复为空链表。成员函数 `insert()`、`_delete()`、`searchByValue()`、`count()`、`display()`、`distinguishing()`、`reverse()` 分别实现了对链表中元素的插入、删除、查找、计数、展示、去重、逆置操作。

第二组:

```

typedef struct term
{
    double coef = 0.0; // coefficient
    int expn = 0;      // exponent
    term() { cin >> coef >> expn; }
    term(double c, int e) : coef(c), expn(e) {}
} ElemType;

typedef struct Node : public ElemType
{
    struct Node *next = nullptr;
    Node() : term() {}
    Node(double c, int e) : term(c, e) {}
    Node(Node &n) : term(n.coef, n.expn) {}
    bool operator==(Node &n) { return expn == n.expn; }
    bool operator<(Node &n) { return expn < n.expn; }
    bool operator>(Node &n) { return expn > n.expn; }
    bool operator==(double d) { return coef == d; }
} * pNode;

typedef struct polynomial
{
    pNode head;
    polynomial(const size_t n) : head(new Node(0.0, 0)) ...
    polynomial() : head(new Node(0.0, 0)) {}
    polynomial(const polynomial &p) : head(p.head) {}
    polynomial(const polynomial *pp) : head(pp ? pp->head : nullptr) {}
    polynomial &clear() ...
    ~polynomial() {}
    polynomial &insert(pNode p_new) ...
    polynomial operator+(const polynomial &p) ...
    polynomial operator*(const polynomial &p) ...
} LinkList;

ostream &operator<<(ostream &out, const polynomial &poly) ...

```

`struct term` 的两个同名构造函数，功能分别是通过由实参传入、或从输入流对象 `cin` 中读入的方式给成员变量 `double coef` 和 `int expn` 赋初值。

`struct Node` 以公有继承的方式继承了 `struct term`，并且增加了成员变量 `Node *next`，用于存储指向下一结点的指针。3 个构造函数分别实现了构造空结点、由实参传入、或从输入流对象 `cin` 中读入的方式给各成员变量赋初值。对运算符 `==`、`<`、`>` 的重载函数分别返回指数的大小比较结果，使两个结点的指数 `expn` 更易比较。

`struct polynomial` 有一个成员变量 `pNode head`，存储指向该多项式链表的头指针。各构造函数分别实现了从标准输入流中读入 `n` 组数据、不读数据、由现有多项式克隆构造新的多项式链表。成员函数 `clear()` 实现了清除多项式中所有元素的功能；`insert()` 函数实现了多项式的项的插入，并且能够处理相同指数项的合并、系数为 `0` 的结点的清除，避免出现同类项未合并、为 `0` 的项未删除等情况。对运算符 `+` 和 `*` 的重载分别实现了多项式的加法和乘法。

开发环境 Windows 10, Visual Studio Code with g++, C++ language

第一组:

```
*****
*   Linked List Conductor   *
*****
Command list:
1> Create a linked list
2> Display the linked list
3> Insert an element
4> Delete an element
5> Search for an element
6> Distinguish the linked list
7> Reverse the linked list
0> Exit

Please enter your command:
```

此为主菜单。

```
Please enter your command: 2
Input illegal. Please try again.

Press any key to continue...
```

若未创建链表就进行除创建和退出的其他操作，直接抛出错误提示信息，并重新进入菜单。

```
Please enter your command: 1
Please input the number of elements:
8
Please input the elements relatively:
2 4 7 2 3 5 7 8
The linked list is created successfully!

Press any key to continue...
```

成功以尾插法创建链表。链表元素包括两组重复数据（2 和 7）。

```
Please enter your command: 3
Please input the position and the value of the inserted element:
89 0
Illegal. Please try again.
```

插入的位置为 0 或超出当前链表长度时，抛出非法提示信息。

```
Please enter your command: 3
Please input the position and the value of the inserted element:
3
6
The element inserted successfully.
```

插入成功。

```
Please enter your command: 2
Elements of the linked list are as below:
8 7 6 5 3 2 7 4 2
There are 9 elements in the linked list.
```

展示链表内所有元素。

```
Please enter your command: 4
Please input the position of the element to be deleted:
8
The element deleted successfully.
```

删除元素。

```
Please enter your command: 2
Elements of the linked list are as below:
8 7 6 5 3 2 7 2
There are 8 elements in the linked list.
```

观察到成功删除。

```
Please enter your command: 6
The linked list is successfully distinguished.

去重。
```

调试分析

```
Please enter your command: 2
Elements of the linked list are as below:
8 7 6 5 3 2
There are 6 elements in the linked list.
```

观察到成功去重。

```
Please enter your command: 5
Please input the value of the element to be searched for:
8
Found! The element's position is 1.
```

按值查找元素，找到并反馈其在链表中的顺序。

```
Please enter your command: 5
Please input the value of the element to be searched for:
888
Not found.
```

按值查找元素，提示未找到。

```
Please enter your command: 7
Please input the start and the end of the part of the list to be reversed:
3 8
Input illegal. Please try again.
```

输入需要逆置部分的首尾序号非法。

```
Please enter your command: 7
Please input the start and the end of the part of the list to be reversed:
2 5
The list is successfully reversed.
```

逆置 2 至 5 结点。

```
Please enter your command: 2
Elements of the linked list are as below:
8 3 5 6 7 2
There are 6 elements in the linked list.
```

观察到成功逆置。

```
*****
*   Linked List Conductor   *
*****
Command list:
1> Create a linked list
2> Display the linked list
3> Insert an element
4> Delete an element
5> Search for an element
6> Distinguish the linked list
7> Reverse the linked list
0> Exit

Please enter your command: 0
Program ending...
```

退出程序。

第二组：

```
*****
*   Polynomial Calculator   *
*****

Welcome! This program is committed to conducting calculations of two polynomials.
Please input the number of terms of the first polynomial:
8
Please input coefficient and exponent of each term relatively:
-5 2 4 68 9 0 23 1 5 7 3 2 8 4 3 45
Please input the number of terms of the second polynomial:
6
Please input coefficient and exponent of each term relatively:
-4 6 5 2 6 45 2 6 -7 8 3 24 _
```

输入数据。所输入多项式的各项为乱序，并且含有指数相同的项。

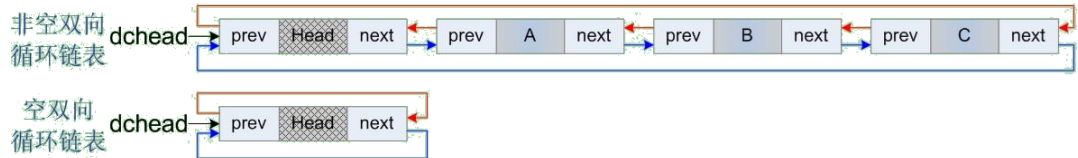
	<div data-bbox="352 194 780 546"><pre>***** * Polynomial Calculator * ***** Input successfully! The first polynomial is: +9x^0+23x^1-2x^2+8x^4+5x^7+3x^45+4x^68 The second polynomial is: +5x^2-2x^6-7x^8+3x^24+6x^45 Command list: 0> add 1> multiple 2> add and multiple Please input your choice: _</pre></div> <p>展示所输入多项式，发现同类项被合并，并且系数为 0 的项被删去。提示命令列表。</p> <div data-bbox="352 618 713 777"><pre>Command list: 0> add 1> multiple 2> add and multiple Please input your choice: 6 Input illegal. Please try again.</pre></div> <p>非法输入。</p> <div data-bbox="352 851 1377 1057"><pre>Command list: 0> add 1> multiple 2> add and multiple Please input your choice: 2 The result of two polynomials added is: +9x^0+23x^1+3x^2+8x^4-2x^6+5x^7-7x^8+3x^24+9x^45+4x^68 The result of two polynomials multiplied is: +45x^2+115x^3-10x^4+22x^6-46x^7-59x^8-136x^9-2x^10-56x^12-10x^13-35x^15+27x^24+69x^25-6x^26+24x^28+15x^31+54x^45+138x^46+3x^47+48x^49-6x^51+30x^52-21x^53+9x^69+20x^70-8x^74-28x^76+18x^90+12x^92+24x^113</pre></div> <p>输出结果并结束程序。</p> <p>综上，两组实验都很好地完成了题目对单向链表功能的要求，功能完好，使用方便，并且能够较好地处理非法数据的输入并反馈相关错误提示，程序界面友好，具有比较恰当的人性化提醒，具有一定的鲁棒性。</p>
<p>心得体会</p>	<p>一、实验总结</p> <p>在本次上机实验中，我复习了理论课上学到的线性表的定义和作用，将课本上线性表的链表存储结构和与其相关的建立、遍历、查找、插入、删除和去重等函数样例封装成了 struct LinkList 结构体。</p> <p>链表的优点有，插入删除速度快，内存利用率高，不会浪费内存；大小没有固定，拓展很灵活。链表的缺点有不能随机查找，必须从第一个开始遍历，查找效率低。</p> <p>从存储空间利用率角度出发：顺序表的存储空间是静态分配的，在执行程序前需预先分配一定长度的连续的存储空间。当线性表的长度 n 不能预先确定时，可能会分配的过大或过小，则导致存储空间的浪费或溢出；链表的存储空间是动态分配的，无需预先分配存储空间，只要内存中尚有可分配空间就不会发生溢出。但指针域需占额外的存储空间。所以当线性表长度变化较大或难以估计时，宜采用链表存储结构；当线性表长度变化不大，且能预先估计存储量小时，为节省存储空间，宜采用顺序表存储结构。</p> <p>从时间效率角度考虑：若对线性表进行的主要操作是查找，很少进行插入和删除操作时，宜采用顺序表存储结构；若对线性表进行插入和删除操作频繁时，宜采用链表作为存储结构。</p> <p>二、双向循环链表的存储结构描述</p> <p>双向链表通常采用带表头结点的循环链表形式，即双向循环链表。双向循环链表在双向链表的基础上，将表头结点的前驱指针指向尾结点，尾结点的后驱指针指向头结点，首</p>

尾相连形成一个双向环。双向循环链表可方便地获取当前结点的前驱结点，不必像单向循环链表那样从头开始遍历；而其循环的特性又可方便地从任一结点出发单向遍历整个链表，不必像双向链表那样根据方向而使用不同的指针域。

带头结点的双向循环链表如下图所示：

```
typedef struct dNode
{
    ElemType data = 0;
    dNode *prior = nullptr;
    dNode *next = nullptr;
} * node_ptr; // struct dNode

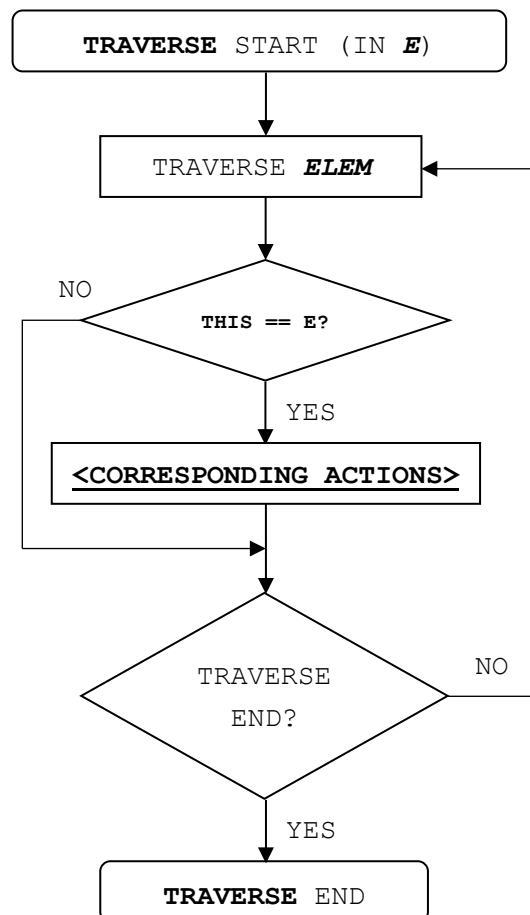
struct dLinkList
{
    node_ptr head;
    node_ptr tail;
    size_t length = 0;
}; // struct dLinkList
```



三、性能分析

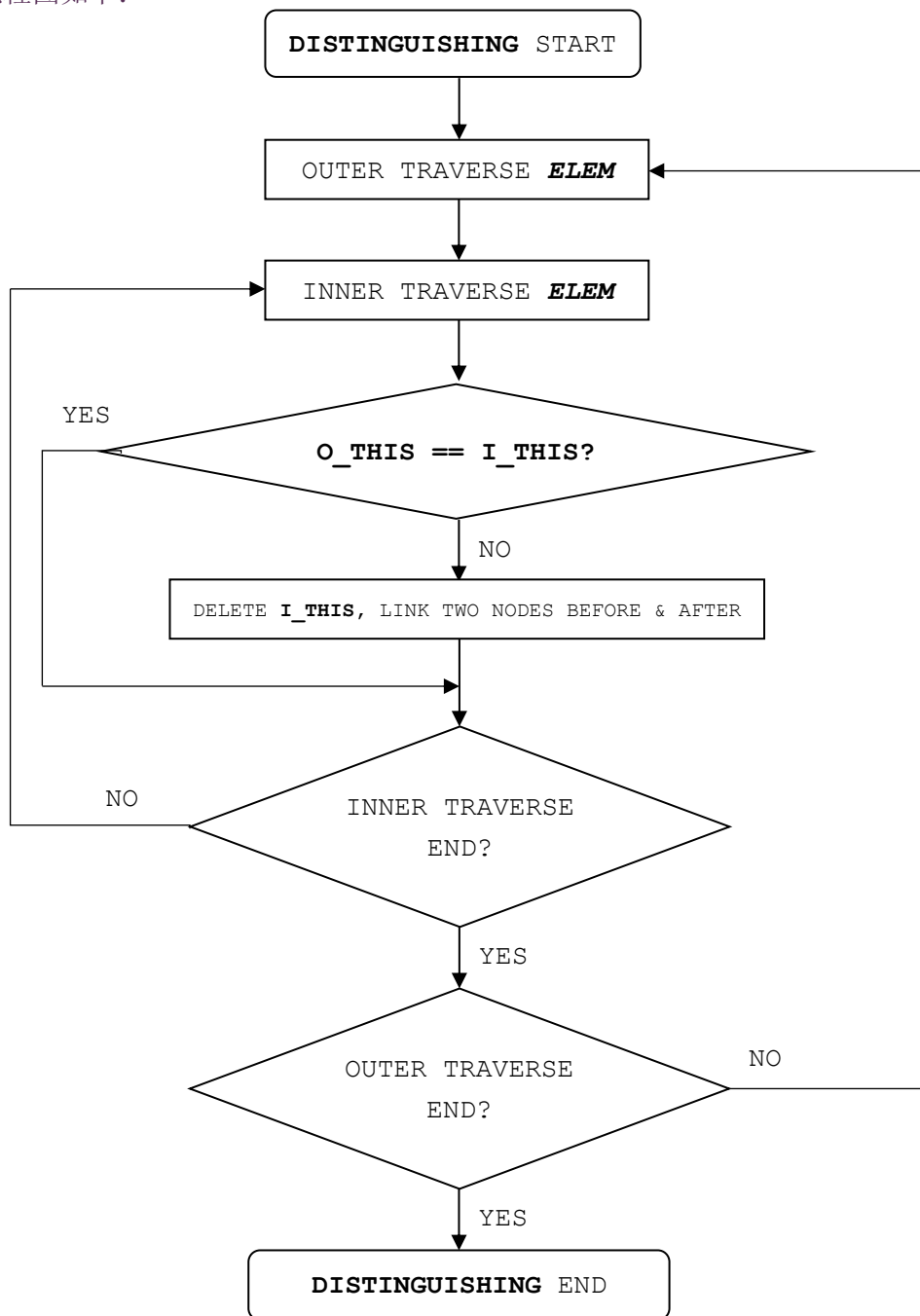
- (1) 链表的遍历相关操作（建立、查找、插入、删除等操作）
时间复杂度为 $O(n)$ 。

流程图如下：



- (2) 链表的去重

时间复杂度为 $O(n^2)$ ，空间复杂度为 $O(1)$ 。
流程图如下：



(3) 链表的逆置

时间复杂度为 $O(n)$ ，空间复杂度为 $O(1)$ 。
流程图如下：

	<pre>graph TD; A[REVERSE START (IN O, D)] --> B[TRAVERSE LINKLIST AND RECORD O- AND D-CORRESPONDING NODES]; B --> C[TRAVERSE LINKLIST BETWEEN O_TH & D_TH]; C --> D["<u>REVERSING ACTIONS</u> TMP = THIS->NEXT; THIS->NEXT = TMP_PRE; TMP_PRE = THIS; THIS = TMP;"]; D --> E{TRAVERSE END?}; E -- NO --> C; E -- YES --> F[REVERSE END];</pre>
第一组 源码	<pre>1. #ifdef __GNUC__ 2. #include <bits/stdc++.h> 3. #endif 4. 5. #ifdef _MSC_VER 6. #define _CRT_SECURE_NO_WARNINGS 7. #include <iostream> 8. #include <cstdlib> 9. #include <cstring> 10. #include <cmath> 11. #endif 12. 13. #include <windows.h> 14. #include <conio.h> 15. 16. using namespace std; 17. 18. #define TRUE 1</pre>

```

19. #define FALSE 0
20. #define OK 1
21. #define ERROR 0
22. #define INFEASIBLE -1
23. typedef int Status;
24. typedef int Boolean;
25. typedef int ElemType;
26.
27. typedef struct Node
28. {
29.     ElemType data = 0;
30.     struct Node *next = nullptr;
31.     Node() { cin >> data; }
32.     Node(const ElemType &e) : data(e) {}
33. } * pNode;
34.
35. struct LinkList
36. {
37.     pNode head = nullptr;
38.     LinkList(const size_t n) : head(new Node(0))
39.     { // create a linked list through head-inserted method
40.         pNode p = nullptr;
41.         for (size_t i = n; i > 0; i--)
42.         {
43.             p = new Node();
44.             p->next = head->next;
45.             head->next = p;
46.         }
47.     }
48.     LinkList(const size_t n, int) : head(new Node(0))
49.     { // create a linked list through tail-inserted method
50.         pNode p = head;
51.         for (size_t i = 0; i < n; i++)
52.         {
53.             p->next = new Node();
54.             p = p->next;
55.         }
56.     }
57.     ~LinkList() {}
58.     LinkList &clear()
59.     { // destroy the linked list
60.         while (head)
61.         {
62.             pNode p = head->next;

```

```

63.         delete head;
64.         head = p;
65.     }
66.     return *this;
67. }
68. Status insert(const size_t n, const ElemType &e)
69. { // insert a Node whose value is e, before n_th position
70.     pNode p = head, q = nullptr;
71.     size_t i = 0;
72.     for (; i < n - 1 && p; i++)
73.     {
74.         p = p->next;
75.     }
76.     if (i > n - 1 || !p)
77.     {
78.         return ERROR;
79.     }
80.     q = new Node(e);
81.     q->next = p->next;
82.     p->next = q;
83.     return OK;
84. }
85. Status _delete(const size_t n)
86. { // delete n_th Node
87.     pNode p = head, q = nullptr;
88.     size_t i = 0;
89.     for (; i < n - 1 && p->next; i++)
90.     {
91.         p = p->next;
92.     }
93.     if (i > n - 1 || !(p->next))
94.     {
95.         return ERROR;
96.     }
97.     q = p->next;
98.     p->next = q->next;
99.     delete q;
100.    return OK;
101. }
102. Status searchByValue(const ElemType &e)
103. {
104.     pNode p = head->next;
105.     for (int i = 1; p; i++)
106.     {

```

```

107.         if (p->data == e)
108.         {
109.             return i;
110.         }
111.         p = p->next;
112.     }
113.     return INFEASIBLE;
114. }
115. size_t count()
116. {
117.     size_t n = 0;
118.     pNode p = head->next;
119.     while (p)
120.     {
121.         n++;
122.         p = p->next;
123.     }
124.     return n;
125. }
126. size_t display()
127. {
128.     pNode p = head->next;
129.     size_t _count = 0;
130.     while (p)
131.     {
132.         cout << p->data << " ";
133.         p = p->next;
134.         _count++;
135.     }
136.     cout << endl;
137.     return _count;
138. }
139. Status distinguishing()
140. {
141.     pNode p = head->next, q, k;
142.     while (p->next)
143.     {
144.         q = p;
145.         k = q->next;
146.         while (k)
147.         {
148.             if (p->data == k->data)
149.             {
150.                 q->next = k->next;

```

```

151.         delete k;
152.         k = q->next;
153.     }
154.     else
155.     {
156.         q = k;
157.         k = k->next;
158.     }
159. }
160. if (p->next)
161. {
162.     p = p->next;
163. }
164. }
165. return OK;
166. }
167. Status reverse(const size_t O, const size_t D)
168. { // reverse the linked list from O_th to D_th elements
169.     if (O < 1U || O >= D)
170.     {
171.         return ERROR;
172.     }
173.     size_t len = 0;
174.     pNode cur = head->next, o = nullptr, o_pre = nullptr, d_next = nullptr;
175.
176.     while (cur)
177.     {
178.         ++len;
179.         if (len == O - 1)
180.         {
181.             o_pre = cur;
182.         }
183.         if (len == D + 1)
184.         {
185.             d_next = cur;
186.         }
187.         cur = cur->next;
188.     }
189.     if (len < D)
190.     {
191.         return ERROR;
192.     }
193.     o = o_pre ? o_pre->next : head->next;
194.

```

```

195.         pNode tmp_pre = d_next, tmp = nullptr;
196.
197.         while (o != d_next)
198.         {
199.             tmp = o->next;
200.             o->next = tmp_pre;
201.             tmp_pre = o;
202.             o = tmp;
203.         }
204.
205.         if (o_pre)
206.         {
207.             o_pre->next = tmp_pre;
208.         }
209.         else
210.         {
211.             head->next = tmp_pre;
212.         }
213.         return OK;
214.     }
215. }; // struct LinkList
216.
217. inline int menu()
218. {
219.     system("cls");
220.     cout << "*****" << endl;
221.     cout << "*   Linked List Conductor   *" << endl;
222.     cout << "*****" << endl;
223.     cout << "Command list: " << endl;
224.     cout << "    1> Create a linked list" << endl;
225.     cout << "    2> Display the linked list" << endl;
226.     cout << "    3> Insert an element" << endl;
227.     cout << "    4> Delete an element" << endl;
228.     cout << "    5> Search for an element" << endl;
229.     cout << "    6> Distinguish the linked list" << endl;
230.     cout << "    7> Reverse the linked list" << endl;
231.     cout << "    0> Exit" << endl;
232.     cout << endl;
233.     cout << "Please enter your command: ";
234.     return _getche() - '0';
235. }
236.
237. int main()
238. {

```

```

239.  size_t n = 0, position = 0;
240.  size_t left = 0, right = 0;
241.  int order = 0;
242.  ElemType value;
243.  LinkList *L = nullptr;
244.
245.  while (true)
246.  {
247.      switch (order = menu())
248.      {
249.          case 0: // exit
250.              cout << endl
251.                  << endl
252.                  << "Program ending..." << endl;
253.              if (L)
254.              {
255.                  L->clear();
256.                  delete L;
257.              }
258.              Sleep(1000);
259.              exit(0);
260.              break;
261.          case 1: // create
262.              cout << endl
263.                  << endl;
264.              if (L)
265.              {
266.                  delete L;
267.                  L = nullptr;
268.              }
269.              cout << "Please input the number of elements: " << endl;
270.              cin >> n;
271.              cout << "Please input the elements relatively: " << endl;
272.              L = new LinkList(n);
273.              cout << "The linked list is created successfully! " << endl;
274.              break;
275.          default:
276.              if (L)
277.              {
278.                  switch (order)
279.                  {
280.                      case 2: // display
281.                          cout << endl
282.                              << endl

```

```

283.             << "Elements of the linked list are as below: " << endl;
284.             n = L->display();
285.             cout << "There are " << n << " elements in the linked list. " << endl;
286.             break;
287.         case 3: // insert
288.             cout << endl
289.                 << endl
290.                 << "Please input the position and the value of the inserted element: "
291.                 << endl;
292.             cin >> position >> value;
293.             if (L->insert(position, value))
294.             {
295.                 cout << "The element inserted successfully. " << endl;
296.             }
297.             else
298.             {
299.                 cerr << "Illegal. Please try again. " << endl;
300.             }
301.             break;
302.         case 4: // delete
303.             cout << endl
304.                 << endl
305.                 << "Please input the position of the element to be deleted: " << endl;
306.             cin >> position;
307.             if (L->_delete(position))
308.             {
309.                 cout << "The element deleted successfully. " << endl;
310.             }
311.             else
312.             {
313.                 cout << "Illegal. Please try again. " << endl;
314.             }
315.             break;
316.         case 5: // search
317.             cout << endl
318.                 << endl
319.                 << "Please input the value of the element to be searched for:" << endl
320.                 << endl;
321.             cin >> value;
322.             position = L->searchByValue(value);
323.             if (position == (size_t)INFEASIBLE)
324.             {
325.                 cerr << "Not found. " << endl;

```



```

324.         }
325.         else
326.         {
327.             cout << "Found! The element's position is " << position << "." << endl;
328.         }
329.         break;
330.         case 6: // distinguishing
331.             cout << endl
332.                 << endl;
333.             L->distinguishing();
334.             cout << "The linked list is successfully distinguished. " << endl;
335.             break;
336.         case 7: // reverse
337.             cout << endl
338.                 << endl
339.                 << "Please input the start and the end of the part of the list to be r
eversted: " << endl;
340.             cin >> left >> right;
341.             if (L->reverse(left, right))
342.             {
343.                 cout << "The list is successfully reversed. " << endl;
344.             }
345.             else
346.             {
347.                 cerr << "Input illegal. Please try again. " << endl;
348.             }
349.             break;
350.         }
351.     }
352.     else
353.     {
354.         cout << endl
355.             << endl;
356.         cerr << "Input illegal. Please try again." << endl;
357.     }
358.     break;
359. }
360. cout << endl
361.     << endl
362.     << "Press any key to continue..." << endl;
363.     (void)_getch();
364. }
365.

```

	<pre> 366. if (L) 367. { 368. L->clear(); 369. delete L; 370. } 371. 372. return 0; 373. }</pre>
<p>第二组 源码</p>	<pre> 1. #ifdef __GNUC__ 2. #include <bits/stdc++.h> 3. #endif 4. 5. #ifdef _MSC_VER 6. #define _CRT_SECURE_NO_WARNINGS 7. #include <iostream> 8. #include <cstdlib> 9. #include <cstring> 10. #include <cmath> 11. #endif 12. 13. #include <windows.h> 14. #include <conio.h> 15. 16. using namespace std; 17. 18. #define TRUE 1 19. #define FALSE 0 20. #define OK 1 21. #define ERROR 0 22. #define INFEASIBLE -1 23. typedef int Status; 24. typedef int Boolean; 25. typedef struct term 26. { 27. double coef = 0.0; // coefficient 28. int expn = 0; // exponent 29. term() { cin >> coef >> expn; } 30. term(const double c, const int e) : coef(c), expn(e) {} 31. } ElemType; 32. 33. typedef struct Node : public ElemType 34. {</pre>

```

35.     struct Node *next = nullptr;
36.     Node() : term() {}
37.     Node(const double c, const int e) : term(c, e) {}
38.     Node(const Node &n) : term(n.coef, n.expn) {}
39.     bool operator==(const Node &n) { return expn == n.expn; }
40.     bool operator<(const Node &n) { return expn < n.expn; }
41.     bool operator>(const Node &n) { return expn > n.expn; }
42.     bool operator==(const double d) { return coef == d; }
43. } * pNode;
44.
45. typedef struct polynomial
46. {
47.     pNode head;
48.     polynomial(const size_t n) : head(new Node(0.0, 0))
49.     {
50.         for (size_t i = 0; i < n; i++)
51.         {
52.             pNode t = new Node();
53.             insert(t);
54.         }
55.     }
56.     polynomial() : head(new Node(0.0, 0)) {}
57.     polynomial(const polynomial &p) : head(p.head) {}
58.     polynomial(const polynomial *pp) : head(pp ? pp->head : nullptr) {}
59.     polynomial &clear()
60.     { // clear all elements and recover to an empty linked list
61.         while (head)
62.         {
63.             pNode p = head->next;
64.             delete head;
65.             head = p;
66.         }
67.         return *this;
68.     }
69.     ~polynomial() {}
70.     polynomial &insert(pNode p_new)
71.     { // insert an element into the linked list on the existed rule
72.         pNode m = head, n = head->next;
73.
74.         do
75.         {
76.             if (!p_new)
77.             {
78.                 break;

```

```

79.         }
80.         if (!(p_new->coef))
81.         {
82.             delete p_new;
83.             break;
84.         }
85.         if (!head->next)
86.         {
87.             head->next = p_new;
88.         }
89.         else
90.         {
91.             while (true)
92.             {
93.                 if (!n->next && *n < *p_new)
94.                 {
95.                     n->next = p_new;
96.                     break;
97.                 }
98.                 else if (*n > *p_new)
99.                 {
100.                    m->next = p_new;
101.                    p_new->next = n;
102.                    break;
103.                }
104.                else if (*n == *p_new)
105.                {
106.                    n->coef += p_new->coef;
107.                    delete p_new;
108.                    break;
109.                }
110.                else
111.                {
112.                    m = n;
113.                    n = n->next;
114.                }
115.            }
116.        }
117.    } while (false);
118.    return *this;
119. }
120. polynomial operator+(const polynomial &p)
121. {
122.     polynomial result;

```

```

123.     pNode pLeft = this->head->next, pRight = p.head->next, pp = nullptr;
124.     while (pLeft && pRight)
125.     {
126.         if (*pLeft == *pRight)
127.         {
128.             pp = new Node(pLeft->coef + pRight->coef, pLeft->expn);
129.             pLeft = pLeft->next;
130.             pRight = pRight->next;
131.         }
132.         else if (*pLeft < *pRight)
133.         {
134.             pp = new Node(*pLeft);
135.             pLeft = pLeft->next;
136.         }
137.         else // if (*pLeft > *pRight)
138.         {
139.             pp = new Node(*pRight);
140.             pRight = pRight->next;
141.         }
142.         result.insert(pp);
143.     }
144.     while (pLeft)
145.     {
146.         pp = new Node(*pLeft);
147.         result.insert(pp);
148.         pLeft = pLeft->next;
149.     }
150.     while (pRight)
151.     {
152.         pp = new Node(*pRight);
153.         result.insert(pRight);
154.         pRight = pRight->next;
155.     }
156.
157.     return result;
158. }
159. polynomial operator*(const polynomial &p)
160. {
161.     polynomial result;
162.     pNode pLeft = this->head->next, pRight = p.head->next;
163.
164.     while (pLeft)
165.     {
166.         pRight = p.head->next;

```

```

167.         while (pRight)
168.         {
169.             pNode p = new Node(pLeft->coef * pRight->coef, pLeft->expn + pRight->expn);
170.             result.insert(p);
171.             pRight = pRight->next;
172.         }
173.         pLeft = pLeft->next;
174.     }
175.     return result;
176. }
177. } LinkList;

178. ostream &operator<<(ostream &out, const polynomial &poly)
179. {
180.     pNode p = poly.head->next;
181.     while (p)
182.     {
183.         out << showpos << p->coef << "x^";
184.         out << noshowpos << p->expn;
185.         p = p->next;
186.     }
187.     return out;
188. }

189.
190. inline void menu()
191. {
192.     system("cls");
193.     cout << "*****" << endl
194.         << "*   Polynomial Calculator   *" << endl
195.         << "*****" << endl
196.         << endl;
197. }

198.
199. int main()
200. {
201.     size_t m, n;
202.     int choice;
203.     menu();
204.     cout << "Welcome! This program is committed to conducting calculations of two polynomials.
205.     " << endl;
206.     cout << "Please input the number of terms of the first polynomial: " << endl;
207.     cin >> m;
208.     cout << "Please input coefficient and exponent of each term relatively: " << endl;
209.     LinkList l_m(m);

```

```

210.
211.     cout << "Please input the number of terms of the second polynomial: " << endl;
212.     cin >> n;
213.     cout << "Please input coefficient and exponent of each term relatively: " << endl;
214.     LinkList l_n(n);
215.     LinkList add(nullptr), mul(nullptr);
216.
217.     bool pass = false;
218.     while (!pass)
219.     {
220.         menu();
221.         cout << "Input successfully! " << endl;
222.         cout << "The first polynomial is: " << endl
223.             << l_m << endl;
224.         cout << "The second polynomial is: " << endl
225.             << l_n << endl;
226.
227.         cout << endl
228.             << "Command list: " << endl
229.             << "    0> add" << endl
230.             << "    1> multiple" << endl
231.             << "    2> add and multiple" << endl
232.             << endl
233.             << "Please input your choice: ";
234.
235.         switch (choice = _getche() - '0')
236.         {
237.             case 0:
238.                 add = l_m + l_n;
239.                 cout << endl
240.                     << "The result of two polynomials added is: " << endl;
241.                 cout << add << endl;
242.                 pass = true;
243.                 break;
244.             case 1:
245.                 mul = l_m * l_n;
246.                 cout << endl
247.                     << "The result of two polynomials multiplied is: " << endl;
248.                 cout << mul << endl;
249.                 pass = true;
250.                 break;
251.             case 2:
252.                 add = l_m + l_n;
253.                 cout << endl

```

```
254.         << "The result of two polynomials added is: " << endl;
255.         cout << add << endl;
256.         mul = l_m * l_n;
257.         cout << endl
258.         << "The result of two polynomials multiplied is: " << endl;
259.         cout << mul << endl;
260.         pass = true;
261.         break;
262.     default:
263.         cerr << endl
264.         << "Input illegal. Please try again. " << endl;
265.         Sleep(1000);
266.     }
267. }
268. add.clear();
269. mul.clear();
270.
271. (void)_getch();
272. return 0;
273. }
```