```cpp
#include<stdio.h>
#include<stdlib.h>

typedef int ElemType;

typedef struct LNode
{
    ElemType data;
    LNode  *next;
}LNode, *LinkList;

enum  Status
{
    OVERFLOW = -1,
    ERROR = 0,
    OK = 1
};

//函数声明
LinkList CreateList_L(int n);
Status   ListInsert_L(LinkList &L, int i, LNode* &q);
Status   ListDelet_L(LinkList &L, int i, LNode *&q);
Status   Reverse_L(LinkList &L);
Status   Divide_L(LinkList &La, LinkList &Lb);
Status   Connect_L(LinkList &La, LinkList &Lb);
void     Print_L(LinkList &L);
void     Destroy_L(LinkList &L);

int main()
{
    int n;
    char chos;
    LinkList La;
    LinkList Lb;

    printf("You will creat a linked list(La) first\n");
    printf("How many elements in La: ");
    scanf_s("%d", &n);
    printf("Please input %d integer numbers(inverted sequence): \n", n);
    La = CreateList_L(n);
    Lb = CreateList_L(0);
    printf("Congratulations! La has been created:\nLa: ");
    Print_L(La);
    while(1)
    {
        printf("1.Reverse La\n2.Divide La to La&Lb\n3.Connect La&Lb to La\n4.Exit
          \n");
        printf("So choose what you want to do now: ");
        scanf_s("%c", &chos);
```

```
49          while(chos == 10)
50          {
51              scanf_s("%c", &chos);
52          }
53
54          switch(chos)
55          {
56              case '1':
57                  {
58                      Reverse_L(La);
59                  }break;
60              case '2':
61                  {
62                      Divide_L(La, Lb);
63                  }break;
64              case '3':
65                  {
66                      Connect_L(La, Lb);
67                  }break;
68              case '4':
69                  {
70                      Destroy_L(La);
71                      Destroy_L(Lb);
72                      exit(0);
73                  }break;
74              default:
75                  {
76                      printf("Error!\n");
77                  }break;
78          }
79          putchar(10);
80      }
81      system("pause");
82      return 0;
83  }
84
85
86  // 逆位序输入n个元素的值，建立带头结点的单链线性表L
87  LinkList CreateList_L(int n)
88  {
89      LNode *p;
90      LinkList L;
91
92      L=(LinkList)malloc(sizeof(LNode));
93      L->next=NULL;
94
95      for (int i=n; i>0; --i)
96      {
97          p=(LinkList)malloc(sizeof(LNode));
```

```
98          scanf_s("%d",&p->data);
99          p->next=L->next; L->next=p;
100     }
101
102     return L;
103 }
104
105 // 在带头结点的单链线性表L中的第i个位置之前插入结点q
106 Status ListInsert_L(LinkList &L, int i, LNode* &q)
107 {
108     LinkList p = L;
109     int count = 0;
110
111     //p指向第i-1个位置
112     while (p && count <i-1)
113     {
114         p = p->next;
115         ++count;
116     }
117     if (!p || count>i-1)
118     {
119         return ERROR;
120     }
121
122     q->next = p->next;
123     p->next = q;
124
125     return OK;
126 }
127
128 //在带头结点的单链表中删除第i个结点,用指针q返回删除的结点
129 Status ListDelet_L(LinkList &L, int i, LNode* &q)
130 {
131     LNode *p = L;
132     int count = 0;
133     while (p->next && count<i-1)
134     {
135         p = p->next;
136         ++count;
137     }
138     if (!(p->next) || count>i-1)
139     {
140         q = NULL;
141         return ERROR;
142     }
143
144     q = p->next;
145     p->next = q->next;
146
```

```
147     return OK;
148 }
149
150 //倒置链表
151 Status Reverse_L(LinkList &L)
152 {
153     if(!L->next)
154     {
155         printf("There is no element in La!\n");
156         return ERROR;
157     }
158
159     LNode *p, *q;
160     p = L->next->next;
161     L->next->next = NULL;
162
163     while(p)
164     {
165         q = p;
166         p = p->next;
167         q->next = L->next;
168         L->next = q;
169     }
170     printf("La has been reversed:\n");
171     printf("La: ");
172     Print_L(L);
173
174     return OK;
175 }
176
177 Status Divide_L(LinkList &La, LinkList &Lb)
178 {
179     if (Lb->next)
180     {
181         printf("Lb must be a empty list!\n");
182         return ERROR;
183     }
184
185     printf("La: ");
186     Print_L(La);
187     putchar(10);
188
189     LNode *p = La->next;
190     LNode *q = NULL;
191     int count = 2;
192
193     while (p)
194     {
195         ListDelet_L(La, count, q);
```

```
196         if(q)   ListInsert_L(Lb, count-1, q);
197
198         ++count;
199         p = p->next;
200     }
201
202     printf("The La have been Divided:\n");
203     printf("La: ");
204     Print_L(La);
205     printf("Lb: ");
206     Print_L(Lb);
207
208     return OK;
209 }
210
211 Status Connect_L(LinkList &La, LinkList &Lb)
212 {
213     if (!Lb->next)
214     {
215         printf("Lb is a empty list!\n");
216         return ERROR;
217     }
218
219     LNode *p = La;
220     while(p->next)
221     {
222         p = p->next;
223     }
224
225     p->next = Lb->next;
226     Lb->next = NULL;
227
228     printf("La&Lb has been connected:");
229     printf("La: ");
230     Print_L(La);
231
232     return OK;
233 }
234
235 void Print_L(LinkList &L)
236 {
237     LNode *p = L;
238     p = p->next;
239
240     while(p)
241     {
242         printf("%d ",p->data);
243         p = p->next;
244     }
```

```
245     putchar(10);
246 }
247
248 void Destroy_L(LinkList &L)
249 {
250     LNode *p, *q;
251     p = L;
252     q = L->next;
253     while(q)
254     {
255         free(p);
256         p = q;
257         q = q->next;
258     }
259     free(p);
260 }
```