线性表

一.什么是线性表

1.定义

由同类型数据元素构成有序序列的线性结构

- 表中元素的个数称为线性表的长度
- 线性表没有元素时，称为空表
- 表起始位置称表头，表结束位置称表尾

2.抽象数据类型的描述

- 类型名称：线性表（List)
- 数据对象集：线性表是n个元素构成的有序序列
- 操作集：线性表L=List:整数i表示位置，元素X∈Element Type,线性表的操作有

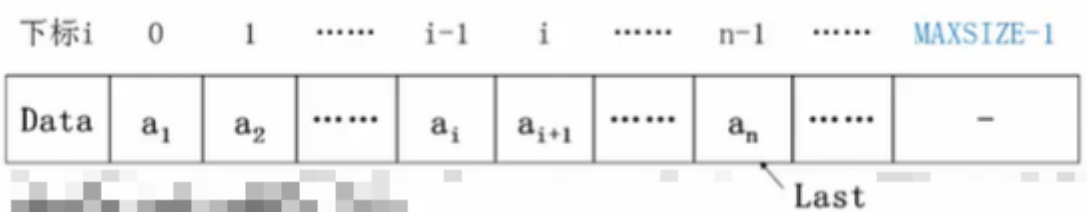
```
List MakeEmpty(); //初始化一个空链表

ElementType FindKth(int K, List L); //根据位序K，返回相应元素

int Find(ElementType X,List L); //在线性表中查找X的第一次出现位置

void Insert(ElementType X,int i,List L); //在位序i前插入一个新元素X;

void Delete(int i,List L); //删除指定位序i的元素

int Length(List L); //返回线性表L的长度n
```

二.线性表的线性存储

1.定义

利用数组的连续存储空间顺序存放线性表的各元素。

2.存储格式



3.代码实现

1.定义表头

```
typedef struct LNode *List;

struct LNode{

    int last; // 线性表最后一个元素的位置
    int data[MaxSize]; // 存储线性表的数据

};

struct LNode L;

List PtrL;
```

2.初始化

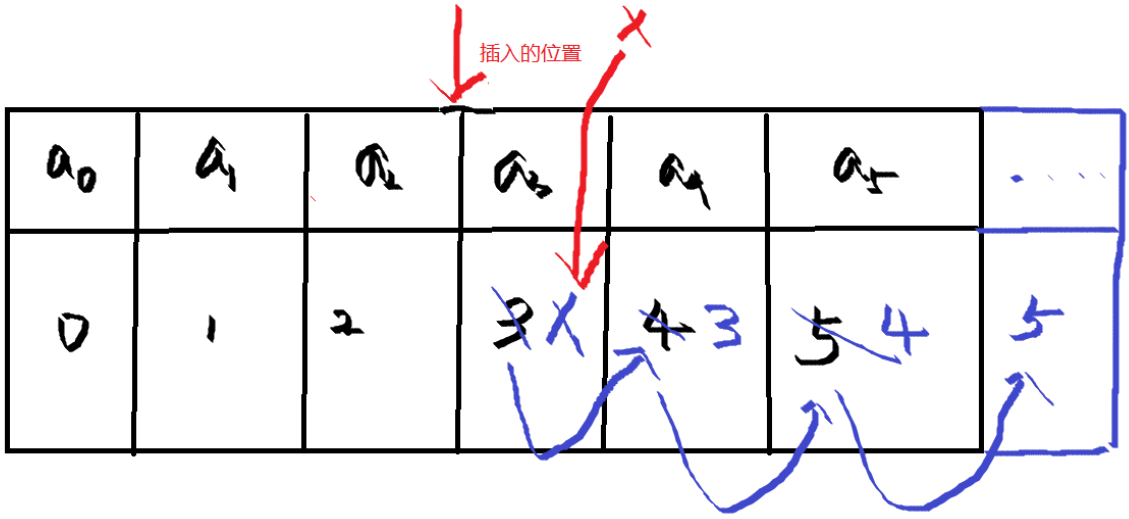
```
//初始化
List MakeEmpty(){

    List PtrL;

    PtrL = (List)malloc(sizeof(struct LNode)); //初始化

    printf("输入表的最后元素的位置\n");

    int length;

    scanf("%d",&length);

    PtrL->last = length;

    return PtrL;

}
```

3.查找

```
//查找
int Find(int X, List PtrL){

    int i = 0;

    /*
    *i <= PtrL->last 防止越界
    *PtrL->data[i] != X 查找
    * */
    while(i <= PtrL->last && PtrL->data[i] != X)
        i++;

    if(i > PtrL->last) //如果没找到，返回-1
        return -1;  //如果找到，返回位序
    else
        return i;

}
```

4.插入

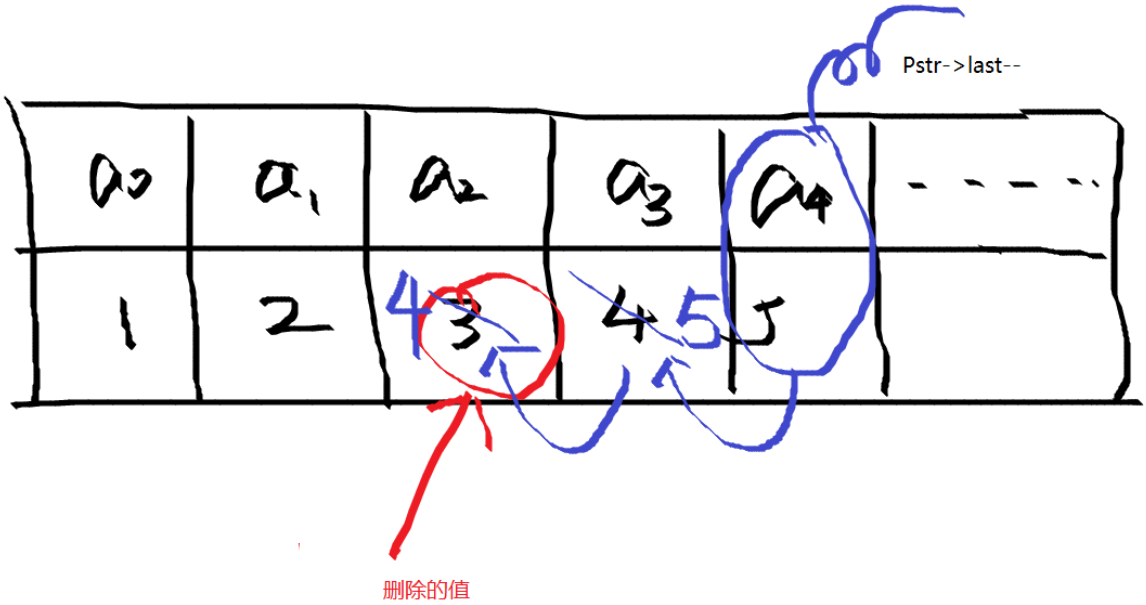
图解




```
//插入
void Insert(int X,int i,List L){/*i代表 a1,a2,....,ai*/

    if(PtrL->last == MaxSize-1){/*判断表是否满*/
        printf("表满");
        return ;
    }

    if(i < 1 || i > PtrL->last+2){/*判断插入位置是否合法*/
        printf("输入的位置不合法\n");
        return ;
    }

    for (int j = PtrL->last; j > i-1; j--)/*数据后移*/
        PtrL->data[j+1] = PtrL->data[j];

    PtrL->data[i] = X;/*插入*/
    PtrL->last++; /*表长加1*/

    return ;

}
```

5.删除

图解




```c
//删除
void Delete(int i,List L){/*i代表 a1,a2,....,ai*/

    if(i < 1 || i > PtrL->last+2){/*判断删除位置是否合法*/
        printf("输入的位置不合法\n");
        return ;
    }

    for (int j = i; j <= PtrL->last; j++)
        PtrL->data[j] = PtrL->data[j+1];

    PtrL->last--;
    return ;

}
```

4.测试代码

```c
#include<stdio.h>
#include<malloc.h>

#define MaxSize 10000

typedef struct LNode *List;

struct LNode{

    int last; // 线性表最后一个元素的位置
    int data[MaxSize]; // 存储线性表的数据

};

struct LNode L;

List PtrL;

//初始化
List MakeEmpty(){
```

```
21
22      List PtrL;
23
24      PtrL = (List)malloc(sizeof(struct LNode)); //初始化
25
26      printf("输入表的最后元素的位置\n");
27
28      int length;
29
30      scanf("%d",&length);
31
32      PtrL->last = length;
33
34      return PtrL;
35
36  }
37
38  //查找
39  int Find(int X, List PtrL){
40
41      int i = 0;
42
43      /*
44      *i <= PtrL->last 防止越界
45      *PtrL->data[i] != X 查找
46      * */
47      while(i <= PtrL->last && PtrL->data[i] != X)
48          i++;
49
50      if(i > PtrL->last) //如果没找到，返回-1
51          return -1;  //如果找到，返回位序
52      else
53          return i;
54
55  }
56
57  //插入
58  void Insert(int X,int i,List L){/*i代表 a1,a2,....,ai*/
59
60      if(PtrL->last == MaxSize-1){/*判断表是否满*/
61          printf("表满");
62          return ;
63      }
64
65      if(i < 1 || i > PtrL->last+2){/*判断插入位置是否合法*/
66          printf("输入的位置不合法\n");
67          return ;
68      }
69
70      for (int j = PtrL->last; j > i-1; j--)/*数据后移*/
71          PtrL->data[j+1] = PtrL->data[j];
72
73      PtrL->data[i] = X;/*插入*/
74      PtrL->last++; /*表长加1*/
75
76      return ;
77
78  }
```

```
79
80  //删除
81  void Delete(int i,List L){/*i代表 a1,a2,....,ai*/
82
83      if(i < 1 || i > PtrL->last+2){/*判断删除位置是否合法*/
84          printf("输入的位置不合法\n");
85          return ;
86      }
87
88      for (int j = i; j <= PtrL->last; j++)
89          PtrL->data[j] = PtrL->data[j+1];
90
91      PtrL->last--;
92      return ;
93
94  }
95
96  //输出表的数据
97  void Display(){
98
99      for (int i = 0; i <= PtrL->last; i++){
100         printf("%d ",PtrL->data[i]);
101     }
102
103     printf("\n");
104
105 }
106
107
108 int main(){
109
110     PtrL = MakeEmpty();
111
112     printf("输入表的数据\n");
113
114     //输入表的值
115     for (int i = 0; i <= PtrL->last; i++){
116
117         int e;
118
119         scanf("%d",&e);
120
121         PtrL->data[i] = e;
122
123    }
124
125     int fx; //查找的值
126
127     printf("请输入查找的元素\n");
128     scanf("%d",&fx);
129
130     printf("查找%d的位置%d\n",fx,Find(fx,PtrL));
131
132     int ix; //插入的值
133     int ii; //插入的位置
134     printf("请输入插入的元素及位置\n");
135     scanf("%d%d",&ix,&ii);
136     Insert(ix,ii,PtrL);
```

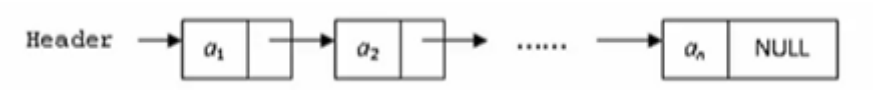
```
137     printf("插入后的表\n");
138     Display();
139 
140     int di; //删除的位置
141     printf("请输入删除的元素的位置\n");
142     scanf("%d",&di);
143     Delete(di,PtrL);
144     printf("删除后的表\n");
145     Display();
146 
147 }
```

三。线性表的链式存储

1.定义

不要求逻辑上相邻的两个元素物理上也相邻；通过“链”建立起数据元素之间的逻辑关系。

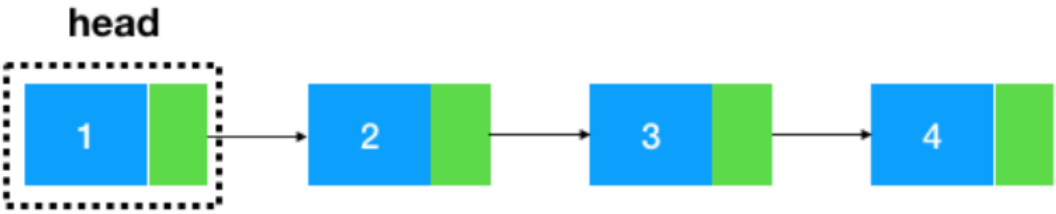
2.存数格式



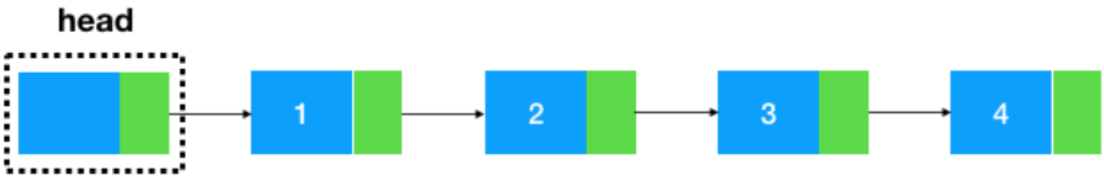
3.前言

1.头指针，头结点的区别

头指针：指向链表第一个的结点



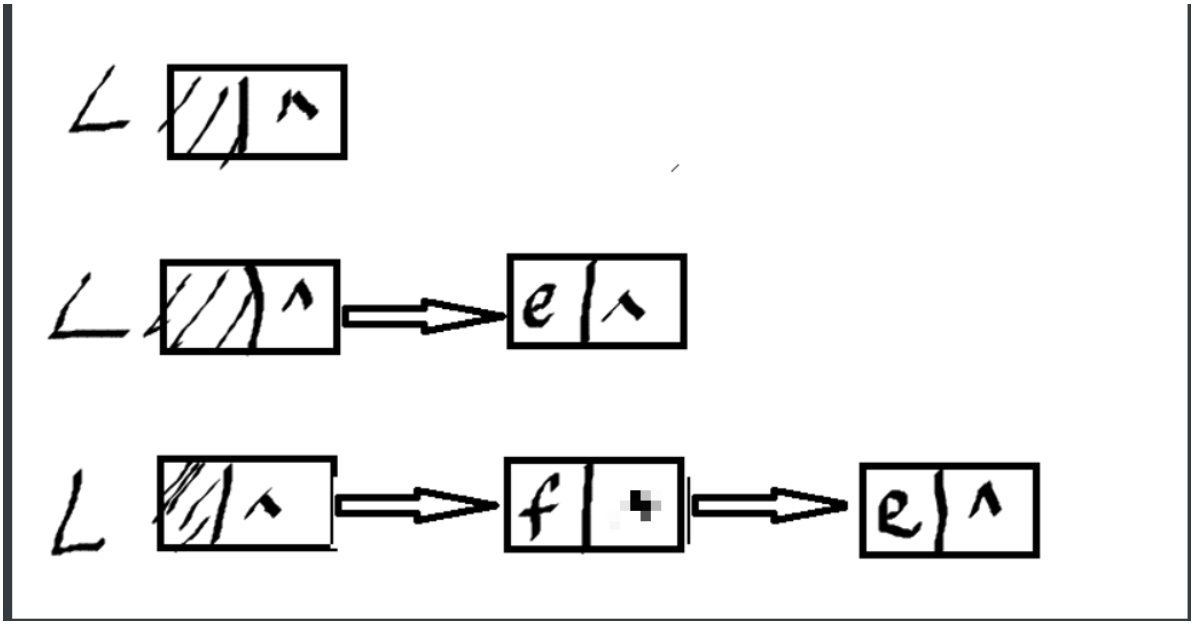
头结点：带头结点的链表的第一个结点，通常结点不存储信息，为虚拟结点。



2.头插法

1.原理

创建一个只有头结点的空链表->生成一个新结点->赋值->插入到头结点之后



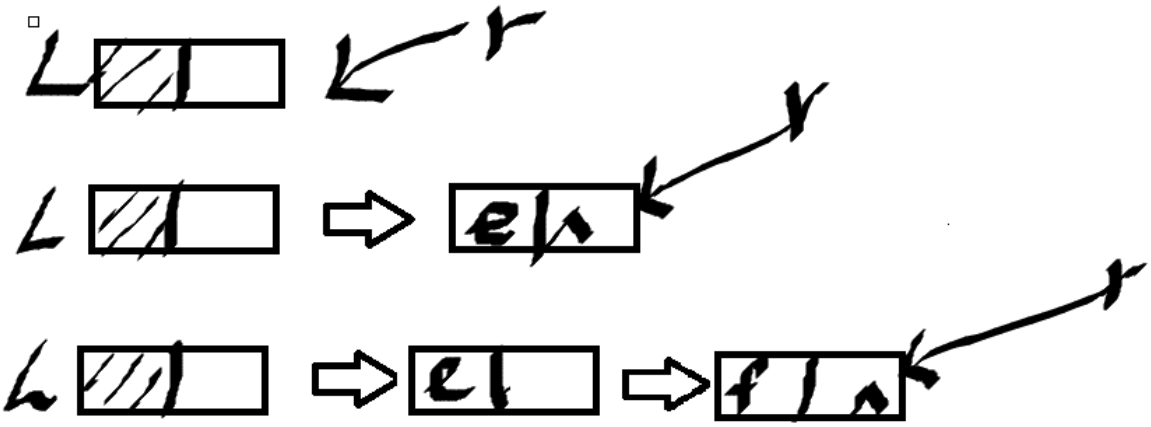
2.代码

```
void CreateList_H(List Ptrl, int length){

    List p;

    for (int i = 0; i < length; i++){

        p = (List)malloc(sizeof(struct LNode));
        scanf("%d",&p->data);
        p->next = Ptrl->next;
        Ptrl->next = p;

    }

}
```

3.尾插法

1.原理

创建一个只有头结点的空链表->尾指针初始化，并指向头结点->生成新的结点->赋值->插入到尾结点->尾指针指向新的尾结点



2.代码

```c
void CreateList_T(List Ptrl, int length){

    List r,p;

    r = Ptrl;

    for (int i = 0; i < length; i++){

        p = (List)malloc(sizeof(struct LNode));
        scanf("%d",&p->data);
        p->next = NULL;
        r->next = p;
        r = p;

    }

}
```

4.代码实现

1.定义表头

```c
typedef struct LNode *List;

struct LNode{

    int data; //存数数据
    List next; //指向下一节点的指针

};

struct LNode L;
List Ptrl;
```

2.初始化

```c
//初始化
List InitList(){

    Ptrl =  (List)malloc(sizeof(struct LNode));
    Ptrl->next = NULL;

    return Ptrl;

}
```

3.按序查找

```c
List FindKth(int K, List Ptrl){

    List p = Ptrl;

    int i = 1;

    while (p && i < K){
```

```
        p = p->next;
        i++;
    }

    if(i == K) return p;
    else return NULL;

}
```

4.按值查找

```
//查找(按值)
List Find(int X, List Ptrl){

    List p = Ptrl;

    while (p != NULL && p->data != X){
        p = p->next;
    }

    return p;

}
```

5.插入

```
/* 插入
1. 用 s 指向一个新的结点
2. 用 p 指向链表的第 i-1 个结点
3. s->Next = p->Next，将 s 的下一个结点指向 p 的下一个结点
4. p->Next = s，将 p 的下一结点改为 s   */
List Insert(int X, int i){/*插入在第i-1个结点*/

    List p, s;

    if( i == 1){ /*新节点插在表头*/

        s = (List)malloc(sizeof(struct LNode)); //申请结点
        s->next = Ptrl; //指向下一个链表
        s->data = X;    //赋值

        return s;

    }

    p = FindKth(i-1,Ptrl); //查找插入位置

    if (p == NULL){

        printf("参数错误");

        return NULL;

    }else{

        s = (List)malloc(sizeof(struct LNode));
```

```
        s->next = p->next;
        s->data = X;
        p->next = s;

        return Ptrl;

    }


}
```

6.删除

```
/* 删除
1. 用 p 指向链表的第 i-1 个结点
2. 用 s 指向要被删除的的第 i 个结点
3. p->Next = s->Next，p 指针指向 s 后面
4. free(s)，释放空间
*/
List Delete(int i,List Ptrl){

    List s,p;

    if(i == 1){

        s = Ptrl;

        if (Ptrl != NULL){
            Ptrl = Ptrl->next;
        }else{
            return NULL;
        }

        free(s);

        return Ptrl;

    }

    p = FindKth(i-1,Ptrl);  //查找删除位置

    if (p == NULL || p->next == NULL){

        printf("节点不存在");

        return NULL;

    }else{

        s = p->next;
        p->next = s->next;
        free(s);

        return Ptrl;
    }

```

```
44
45 }
```

7.测试代码

```
1  #include<stdio.h>
2  #include<malloc.h>
3
4  #define MaxSize 10000
5
6  typedef struct LNode *List;
7
8  struct LNode{
9
10     int data; //存数数据
11     List next; //指向下一节点的指针
12
13 };
14
15 struct LNode L;
16 List Ptrl;
17
18 //初始化
19 List InitList(){
20
21     Ptrl =  (List)malloc(sizeof(struct LNode));
22     Ptrl->next = NULL;
23
24     return Ptrl;
25
26 }
27
28
29 //创建链表
30
31 //方法一：头插法
32 void CreateList_H(List Ptrl, int length){
33
34     List p;
35
36     for (int i = 0; i < length; i++){
37
38         p = (List)malloc(sizeof(struct LNode));
39         scanf("%d",&p->data);
40         p->next = Ptrl->next;
41         Ptrl->next = p;
42
43     }
44
45 }
46
47 //方法二:尾插法
48 void CreateList_T(List Ptrl, int length){
49
50     List r,p;
51
52     r = Ptrl;
```

```
53
54      for (int i = 0; i < length; i++){
55
56          p = (List)malloc(sizeof(struct LNode));
57          scanf("%d",&p->data);
58          p->next = NULL;
59          r->next = p;
60          r = p;
61
62      }
63
64  }
65
66  //求表长
67  int Length(List Ptrl){
68
69      List p = Ptrl;
70
71      int length = 0;
72
73      while (p->next != NULL){
74          p = p->next;
75          length++;
76      }
77
78      return length;
79
80  }
81
82  //查找(序号)
83  List FindKth(int K, List Ptrl){
84
85      List p = Ptrl;
86
87      int i = 1;
88
89      while (p && i < K){
90          p = p->next;
91          i++;
92      }
93
94      if(i == K) return p;
95      else return NULL;
96
97  }
98
99  //查找(按值)
100 List Find(int X, List Ptrl){
101
102     List p = Ptrl;
103
104     while (p != NULL && p->data != X){
105         p = p->next;
106     }
107
108     return p;
109
110 }
```

```
111
112 /* 插入
113 1. 用 s 指向一个新的结点
114 2. 用 p 指向链表的第 i-1 个结点
115 3. s->Next = p->Next，将 s 的下一个结点指向 p 的下一个结点
116 4. p->Next = s，将 p 的下一结点改为 s   */
117 List Insert(int X, int i){/*插入在第i-1个结点*/
118
119     List p, s;
120
121     if( i == 1){ /*新节点插在表头*/
122
123         s = (List)malloc(sizeof(struct LNode)); //申请结点
124         s->next = Ptrl; //指向下一个链表
125         s->data = X;    //赋值
126
127         return s;
128
129     }
130
131     p = FindKth(i-1,Ptrl); //查找插入位置
132
133     if (p == NULL){
134
135         printf("参数错误");
136
137         return NULL;
138
139     }else{
140
141         s = (List)malloc(sizeof(struct LNode));
142
143         s->next = p->next;
144         s->data = X;
145         p->next = s;
146
147         return Ptrl;
148
149     }
150
151
152 }
153
154 /* 删除
155 1. 用 p 指向链表的第 i-1 个结点
156 2. 用 s 指向要被删除的的第 i 个结点
157 3. p->Next = s->Next，p 指针指向 s 后面
158 4. free(s)，释放空间
159 */
160 List Delete(int i,List Ptrl){
161
162     List s,p;
163
164     if(i == 1){
165
166         s = Ptrl;
167
168         if (Ptrl != NULL){
```

```
            Ptrl = Ptrl->next;
        }else{
            return NULL;
        }

        free(s);

        return Ptrl;

    }

    p = FindKth(i-1,Ptrl);

    if (p == NULL || p->next == NULL){

        printf("节点不存在");

        return NULL;

    }else{

        s = p->next;
        p->next = s->next;
        free(s);

        return Ptrl;
    }


}


void Display(List Ptrl){

    List t;
    int flag = 1;

    printf("当前链表为: ");

    for(t = Ptrl->next; t;t = t->next){
        printf("%d  ",t->data);
        flag = 0;
    }

    if(flag)
        printf("NULL");
    printf("\n");
}

int main(){

    Ptrl = InitList();

    int l;
    printf("输入链表初始长度\n");
    scanf("%d",&l);

    //CreateList_H(Ptrl,l);
```

```
    CreateList_T(Ptrl,l);
    Display(Ptrl);

    Insert(4,2);
    Display(Ptrl);

    Delete(2,Ptrl);
    Display(Ptrl);

    return 0;

}
```