```
/*
线性表及其实现
    线性表：
        由同类型数据元素构成有序序列的线性结构

    线性表的链式存储实现
        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

    广义表：
        广义表是线性表的推广
        对于线性表而言，n个元素都是基本的单元素
        广义表中，这些元素不仅可以是单元素，也可以是另一个广义表

    typedef struct GNode *GList;
    struct GNode
    {
        int Tag; // 标志域，0表示结点是单元素，1表示结点是广义表
        union
        {
            ElementType Data;
            GList SubList;
        }URegion; // 子表Sublist与单元素数据域Data复用，共用存储空间
        Glist Next; // 指向后继结点
    }

    多重链表：
        多重链表中结点的指针域会有多个，且链表中的结点可能属于多个链
        包含两个指针域的链表不一定是多重链表，如双向链表不是多重链表
        在树、图中会广泛的用到多重链表

    十字链表：
        存储稀疏矩阵
        结点的数据域：行坐标、列坐标、数值
        每个结点通过两个指针域，把同行、同列串起来
        行指针：Right
        列指针：Down
*/

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

typedef int ElementType; // ElementType 可定义为任意类型
typedef struct LNode *List;
struct LNode{
    ElementType Data; // 数据域
    List Next; // 下一个链表的地址
};
List L;

List MakeEmpty(); // 初始化链表

int Length(List L); // 以遍历链表的方法求链表长度

List FindKth(int K, List L); // 查找链表中第 K 个元素的值

List Find(ElementType X, List L); // 查找链表中值为 X 的结点

List Insert(ElementType X, int i, List L); // 将 X 插入到第 i-1(i>0) 个结点之后

List Delete(int i, List L); // 删除第 i(i>0) 个结点

void Print(List L); // 输出链表元素

// 初始化链表
List MakeEmpty()
{
    List L = (List)malloc(sizeof(struct LNode));
    L = NULL; // L 是一个指向结构体的指针
    return L;
}

// 求表长
```

```
int Length(List L)
{
    List p = L;
    int len = 0;
    while(p) // 当 p 不为空
    {
        len++;
        p = p->Next;
    }
    return len;
}

// 按序查找，设定 K 属于 [1, ...]
List FindKth(int K, List L)
{
    List p = L;
    int i = 1; // 从第 1 个元素开始查找
    while(p && i<K)
    {
        p = p->Next;
        i++;
    }
    if(i == K) // 找到了
        return p;
    else // 未找到
        return NULL;
}

// 查找值为 X 的元素
List Find(ElementType X, List L)
{
    List p = L;
    while(p && p->Data!=X)
        p = p->Next;
    // 找到了，返回 p
    // 未找到，返回 NULL，此时 p 等于 NULL
    return p;
}

/*
插入，将 X 插在位置 i， 设定 i 属于 [1, length+1]
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(ElementType X, int i, List L)
{
    List p, s;
    if(i == 1) // 新结点插入在表头，单独处理
    {
        s = (List)malloc(sizeof(struct LNode));
        s->Data = X;
        s->Next = L;
        return s;
    }
    p = FindKth(i-1, L); // 找到第 i-1 个结点
    if(!p) // 第 i-1 个结点不存在
    {
        printf("结点错误");
        return NULL;
    }
    else
    {
        s = (List)malloc(sizeof(struct LNode));
        s->Data = X;
        s->Next = p->Next; // 将 s 的下一个结点指向 p 的下一个结点
        p->Next = s; // 将 p 的下一结点改为 s
        return L;
    }
}

/*
```

```
/*
删除第 i 个结点，设定 i 属于 [1, n]
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
4. free(s)，释放空间
*/
List Delete(int i, List L)
{
    List p, s;
    if(i==1) // 删除头结点，单独处理
    {
        s = L;
        if(L) // 如果不为空
            L = L->Next;
        else
            return NULL;
        free(s); // 释放被删除结点
        return L;
    }
    p = FindKth(i-1, L); // 查找第 i-1 个结点
    if(!p || !(p->Next)) // 第 i-1 个或第 i 个结点不存在
    {
        printf("结点错误");
        return NULL;
    }
    else
    {
        s = p->Next; // s 指向第 i 个结点
        p->Next = s->Next; // 从链表删除
        free(s); // 释放被删除结点
        return L;
    }
}

// 输出链表元素
void Print(List L)
{
    List t;
    int flag = 1; // 如果链表空，flag == 1
    printf("当前链表为：");
    for(t=L; t; t=t->Next)
    {
        printf("%d  ", t->Data);
        flag = 0;
    }
    if(flag)
        printf("NULL");
    printf("\n");
}

int main()
{
    L = MakeEmpty();
    Print(L);
    L = Insert(11, 1, L);
    L = Insert(25, 1, L);
    L = Insert(33, 2, L);
    L = Insert(77, 3, L);
    Print(L);
    printf("当前链表长度为：%d\n", Length(L));
    printf("此时链表中第二个结点的值是：%d\n", FindKth(2, L)->Data);
    printf("查找22是否在该链表中：");
    if(Find(22, L))
        printf("是！\n");
    else
        printf("否！\n");
    printf("查找33是否在该链表中：");
    if(Find(33, L))
        printf("是！\n");
    else
        printf("否！\n");
    L = Delete(1, L);
    L = Delete(3, L);
```

```
220         printf("----------删除后-----\n");
221         Print(L);
222         return 0;
223     }
224
```