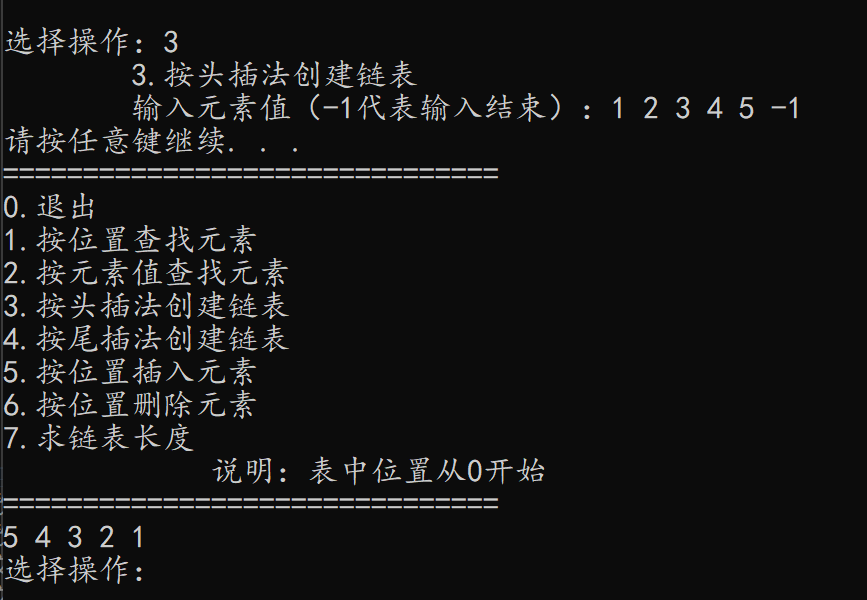
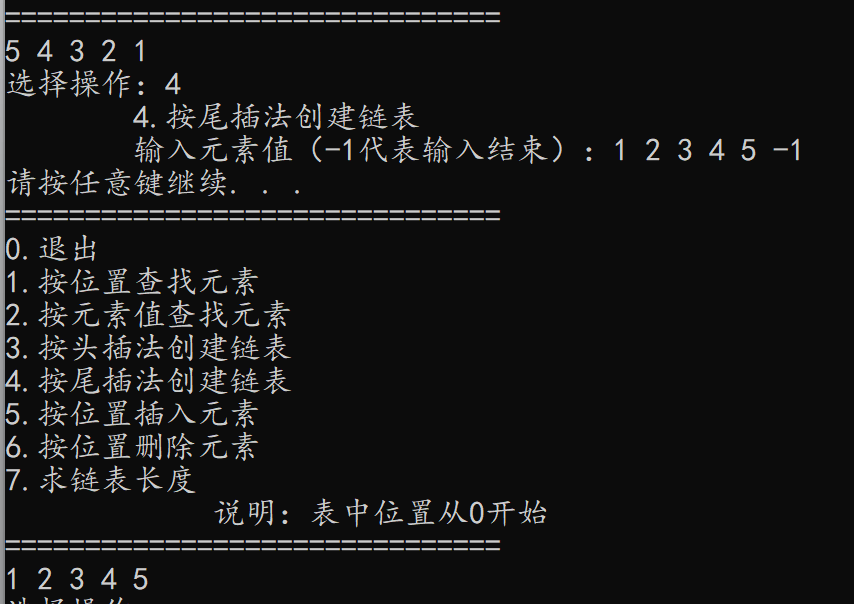
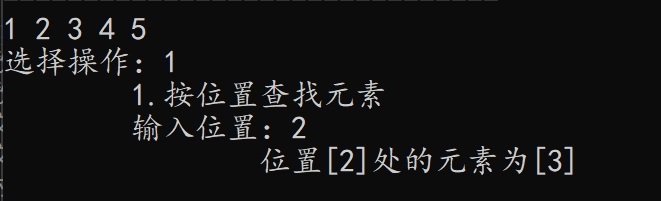
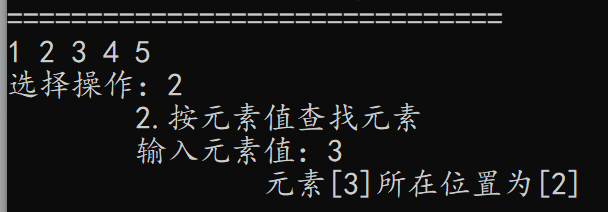
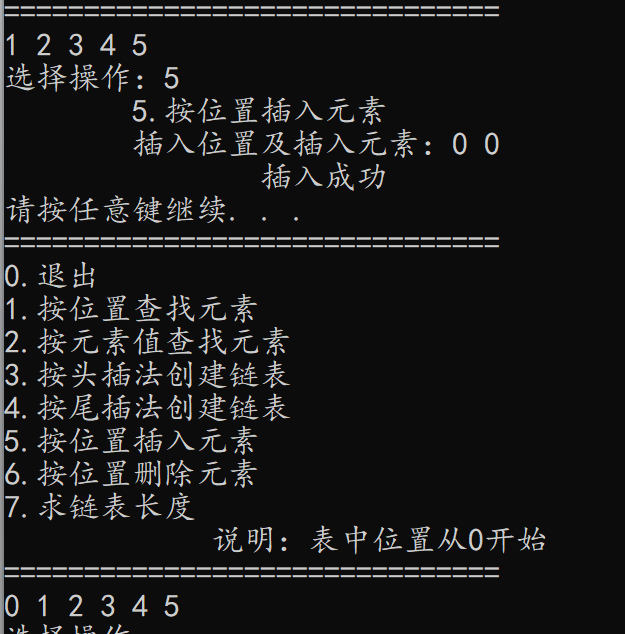
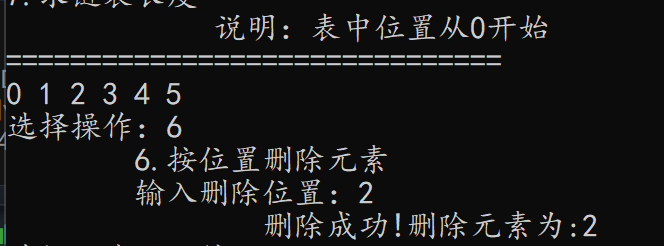
# **单链表**

## 一、程序代码

#include <stdio.h>  
#include <stdlib.h>  
  
#define OK 1  
#define ERROR 0  
//链表的定义  
//结点、链表(指向头结点的一个指针)  
typedef struct Node {  
    int data;  
    struct Node \*next;  
} Node, \*LinkList;  
  
//链表相关操作  
void InitList(LinkList \*L);  
void CreateFromHead(LinkList L);//头插法  
void CreateFromTail(LinkList L);//尾插法  
Node \* Get(LinkList L, int i); //按序号查找  
int LocateLinkList(LinkList L, int key); //按值查找  
int ListLength(LinkList L);//求链表长度  
int InsLinkList(LinkList L, int i, int e); //插入元素  
int DelLinkList(LinkList L, int i, int \*e); //删除元素  
void PrintList(LinkList list);//链表打印  
  
//主函数  
int main() {  
    LinkList L;  
    int sel, i, x;  
    int e;//存储插入的元素以及删除所返回的元素  
    Node \*s;  
  
    InitList(&L);  
  
    while (1) {  
  
        printf("===============================\n");  
        printf("0.退出\n");  
        printf("1.按位置查找元素\n");  
        printf("2.按元素值查找元素\n");  
        printf("3.按头插法创建链表\n");  
        printf("4.按尾插法创建链表\n");  
        printf("5.按位置插入元素\n");  
        printf("6.按位置删除元素\n");  
        printf("7.求链表长度\n");  
        printf("             说明：表中位置从0开始\n");  
        printf("===============================\n");  
        PrintList(L);  
        printf("选择操作：") ;  
        scanf("%d", &sel) ;  
        if (sel == 0) break;  
  
        switch (sel) {  
            case 1:  
                printf("\t1.按位置查找元素\n");  
                printf("\t输入位置：");  
                scanf("%d", &i);  
                s = Get(L, i);  
                if (!s) printf("\t\t位置错误!\n\n");  
                else printf("\t\t位置[%d]处的元素为[%d]\n\n", i, s->data);  
                break;  
            case 2:  
                printf("\t2.按元素值查找元素\n");  
                printf("\t输入元素值：");  
                scanf("%d", &e);  
                i = LocateLinkList(L, e);  
                if (i == -1) printf("\t\t表中没有元素[%d]!\n\n", e);  
                else printf("\t\t元素[%d]所在位置为[%d]\n\n", e, i );  
                break;  
            case 3:  
                printf("\t3.按头插法创建链表\n");  
                printf("\t输入元素值（-1代表输入结束）：");  
                CreateFromHead(L);  
                break;  
            case 4:  
                printf("\t4.按尾插法创建链表\n");  
                printf("\t输入元素值（-1代表输入结束）：");  
                CreateFromTail(L);  
                break;  
            case 5:  
                printf("\t5.按位置插入元素\n");  
                printf("\t插入位置及插入元素：");  
                scanf("%d %d", &i, &e);  
                x = InsLinkList(L, i, e);  
                if (x == ERROR) printf("\t\t插入位置[%d]错误!\n\n", i);  
                else printf("\t\t插入成功\n");  
                break;  
            case 6:  
                printf("\t6.按位置删除元素\n");  
                printf("\t输入删除位置：");  
                scanf("%d", &i);  
                x = DelLinkList(L, i, &e);  
                if (x == ERROR ) printf("\t\t删除位置[%d]错误!\n\n", i);  
                else printf("\t\t删除成功!删除元素为:%d\n", e);  
                break;  
            case 7:  
                printf("\t7.求链表长度\n");  
                x = ListLength(L);  
                printf("\t\t链表长度为:%d\n\n", x);  
                break;  
            default:  
                printf("选项错误！\n");  
                break;  
        }  
        system("pause")    ;  
    }  
}  
  
//建立空链表  
void InitList(LinkList \*L) {  
    \*L = (LinkList)malloc(sizeof(Node));  
    (\*L)->next = NULL; //空链表  
}  
  
void CreateFromHead(LinkList L) {  
    int c;  
    int flag = 1;  
    while (flag) {  
        scanf("%d", &c);  
        if (c != -1) {  
            //创建结点并赋值  
            Node \*s = (Node\*)malloc(sizeof(Node));  
            s->data = c;  
            s->next = L->next; //头插法——在头结点后面插入新结点  
            L->next = s;  
        } else {  
            flag = 0;  
        }  
    }  
}  
  
void CreateFromTail(LinkList L) {  
    Node \*r;//r指向当前链表的最后一个元素  
    int c;  
    int flag = 1;  
    r = L;  
    while (flag) {  
        scanf("%d", &c);  
        if (c != -1) {  
            //创建结点并赋值  
            Node \*s = (Node\*)malloc(sizeof(Node));  
            s->data = c;  
            s->next = NULL;  
            r->next = s;  
            r = r->next; //r后移  
        } else {  
            flag = 0;  
            r->next = NULL;  
        }  
    }  
}  
  
Node \* Get(LinkList L, int i) {  
    int j;  
    Node \*p;  
    if (i < 0) {  
        return NULL;  
    }  
    p = L->next;  
    j = 0;  
    while ((p->next != NULL) && (j < i)) {  
        p = p->next; //p后移一个结点  
        j++;  
    }  
    if (i == j) {  
        return p;  
    } else {  
        return NULL;  
    }  
}  
  
int  LocateLinkList(LinkList L, int key) {  
    Node \*p;  
    p = L->next;  
    int k = 0;  
    while (p != NULL) {  
        if (p->data != key) {  
            p = p->next;  
            k++;  
        } else {  
            break;  
        }  
    }  
    //没找到该元素  
    if (p == NULL) {  
        k = -1;  
    }  
    return k;  
}  
  
int ListLength(LinkList L) {  
    Node \*p;  
    p = L->next;  
    int j = 0;  
    //当p为最后一个结点所指向的下一个结点地址（即NULL）时，循环结束  
    while (p != NULL) {  
        j++;  
        p = p->next;  
    }  
    return j;  
}  
  
int InsLinkList(LinkList L, int i, int e) {  
    Node \*pre = L;//pre指向插入结点的前一个结点  
    int k = 0;  
    if (i < 0) {  
        return ERROR;  
    }  
    //pre最多指向最后一个结点，不能指向最后一个结点所指向的下一个结点的地址（即NULL），因为可能存在在最后一个位置插入元素  
    while (pre != NULL && k <= i - 1) {  
        pre = pre->next;  
        k++;  
    }  
    //pre不能指向最后一个结点  
    if (pre == NULL) {  
        return ERROR;  
    }  
    Node \*s = (Node\*)malloc(sizeof(Node));  
    s->data = e;  
    //插入新结点  
    s->next = pre->next;  
    pre->next = s;  
    return OK;  
}  
  
int DelLinkList(LinkList L, int i, int \*e) {  
    Node \*pre = L, \*r;  
    int k = 0;  
    while (pre->next != NULL && k < i - 1) {  
        pre = pre->next;  
        k++;  
    }  
    //循环结束后，pre指向删除元素的前一个结点，k为i-1  
    //pre不能指向最后一个结点的下一个结点（因为最后一个结点可删除）  
    if (pre->next == NULL) {  
        return ERROR;  
    }  
    r = pre->next;  
    pre->next = r->next;  
    \*e = r->data;  
    free(r);  
    return OK;  
}  
  
//链表打印  
void PrintList(LinkList list) {  
    Node \*p = list->next;//p指向首结点(头结点的下一个元素)  
    while (p != NULL) {  
        printf("%d ", p->data);  
        p = p->next; //p后移  
    }  
    printf("\n");  
    return;  
}

## 运行结果

* 头插法创建链表
  + 
* 尾插法创建链表
  + 
* 按位置查找元素
  + 
* 按元素值查找元素
  + 
* 按位置插入元素
  + 
* 按位置删除元素
  + 
* 求链表长度
  + 