实验2 有序单链表的构建

一、【实验目的】  
1、掌握建立单链表的基本方法。

2、掌握单链表的插入、删除算法的思想和实现

二、【实验内容】

仿照教材中的单链表实现例子，自己设计一个有序单链表，单链表中的数据元素为整型并递增有序。有序单链表的定义:

逻辑结构:有序线性表，数据元素递增有序

存储结构:链式

操作集合:初始化、插入、删除、撤销

(1)ListInitiate(L) 初始化线性表，生成一个空表L。

(2)ListInsert(L,x) 在有序表L中插入数据元素x,使得新表仍然有序。

(3)ListDelete(L,x) 删除有序表L中的数据元素x，若删除成功则返回1，不成功则返回0。

(4)Destroy(L) 撤销单链表

要求:

1.有序单链表的操作集合有如下操作:初始化、插入、删除、撤销，使用头文件单链表的代码。

2.编写主函数main()验证所设计的有序单链表是否能正确插入、删除。

提示:

1.插入操作时，从链表的第一个数据元素结点开始，逐个比较每个结点的data域值和x的值，当data小于等于x时，进行下一个结点的比较；否则就找到了插入结点的合适位置，此时申请新结点把x存入，然后把新结点插入；当比较到最后一个结点仍有data小于等于x时，则把新结点插入单链表尾。

2.删除操作时，从链表的第一个数据元素结点开始，逐个比较每个结点的data域值和x的值，当data不等于x时，进行下一个结点的比较；否则就找到了要删除的结点，删除结点后释放结点。如果到了表尾还没有找到值为x的结点，则链表中没有要删除的元素。

三、实验源代码

/\*单链表结点的结构体定义\*/

typedef struct SingleNode

{

    ElemType data;

    struct SingleNode \*next;

}SingleLinkedList, \*LinkList;

/\*单链表初始化\*/

void ListInitialize(SingleLinkedList \*\*head)

/\*如果有内存空间，申请头结点空间并使头指针head指向头结点\*/

{

    if ((\*head = (SingleLinkedList\*)malloc(sizeof(SingleLinkedList))) == NULL)

        exit(1);

    (\*head)->next = NULL;   /\*置链尾标记NULL\*/

}

/\*求线性表的长度\*/

int ListLength(SingleLinkedList \*head)

{

    SingleLinkedList \*p = head;     /\*p指向头结点\*/

    int size = 0;                   /\*size初始为0\*/

    while (p->next != NULL)         /\*循环计数\*/

    {

        p = p->next;

        size++;

    }

    return size;

}

/\*取数据元素\*/

int ListGet(SingleLinkedList \*head, int i, ElemType \*x)

/\*取序号为i的数据元素的值\*/

{

    SingleLinkedList \*p;

    int j;

    p = head;

    j = -1;

    while (p->next != NULL && j < i)

    {

        p = p->next;

        j++;

    }

    if (j != i)

    {

        printf("Get Error!");

        return 0;

    }

    \*x = p->data;

    return 1;

}

/\*插入数据元素操作\*/

void ListInsert(SingleLinkedList \*head, ElemType x)

{

    SingleLinkedList \*curr, \*pre, \*q;

    pre = head;               /\*pre指向头结点\*/

    curr = head->next;

    while (curr != NULL && curr->data <= x)

    {

        pre = curr;

        curr = curr->next;

    }

    /\*生成新结点由指针q指示\*/

    if ((q = (SingleLinkedList \*)malloc(sizeof(SingleLinkedList))) == NULL)

        exit(1);

    q->data = x;

    q->next = pre->next;      /\*给指针q->next赋值\*/

    pre->next = q;            /\*给指针pre->next赋值\*/

}

/\*删除数据元素x操作\*/

int ListDelete(SingleLinkedList \*head, ElemType x)

{

    SingleLinkedList \*p, \*s;

    p = head;   /\*p指向头结点\*/

    while (p->next != NULL && p->next->data != x ) /\*p指向需删除元素的前一个结点\*/

    {

        p = p->next;

    }

    if (p->next != NULL)

    {

        s = p->next;    /\*指针s指向需删除元素的结点\*/

        p->next = s->next;      /\*删除数据元素x\*/

        free(s);        /\*释放指针s所指向结点的内存空间\*/

        return 1;

    }

    else

    {

        printf("Delete Error!\n");    /\*删除错误返回0\*/

        return 0;

    }

}

/\*单链表释放操作\*/

void Destroy(SingleLinkedList \*\*head)

{

    SingleLinkedList \*p, \*p1;

    p = \*head;

    while (p != NULL)

    {

        p1 = p;

        p = p->next;

        free(p1);

    }

    \*head = NULL;

}

#include <stdio.h>

#include <stdlib.h>

#include <malloc.h>

typedef int ElemType;

#include "Linkedlist.h"

int main(void)

{

    SingleLinkedList \*head;

    int x;

    ListInitialize(&head);  /\*初始化链表\*/

    for (int i = 0; i < 20; i++)

        ListInsert(head, i);    /\*生成一个整型递增有序链表\*/

    printf("Initial list:\n");

    for (int i = 0; i < ListLength(head); i++)      /\*打印初始链表\*/

    {

        ListGet(head, i, &x);

        printf("%d ", x);

    }

    printf("\n");

    printf("\nEnter delete number:");     /\*输入需要删除的元素\*/

    scanf("%d", &x);

    ListDelete(head, x);

    printf("After delete list:\n");     /\*打印删除后的链表\*/

    for (int i = 0; i < ListLength(head); i++)

    {

        ListGet(head, i, &x);

        printf("%d ", x);

    }

    printf("\n");

    printf("\nEnter insert number:");       /\*输入需要插入的元素\*/

    scanf("%d", &x);

    ListInsert(head, x);

    printf("After insert list:\n");

    for (int i = 0; i < ListLength(head); i++)      /\*打印插入后的链表\*/

    {

        ListGet(head, i, &x);

        printf("%d ", x);

    }

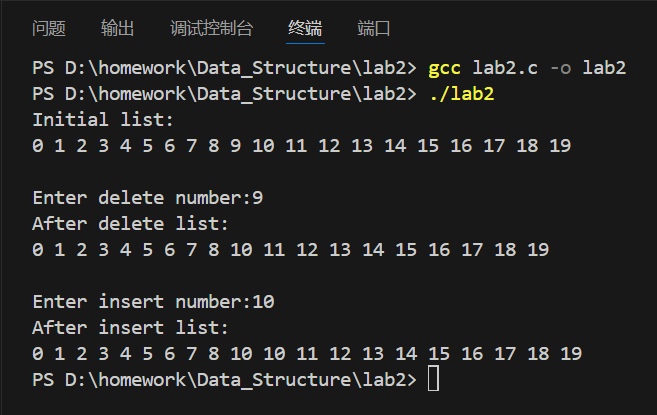
    Destroy(&head);     /\*释放内存\*/

    return 0;

}

四、实验结果

-->



五、实验心得

答：本次实验中，我掌握了建立单链表的基本方法，掌握了单链表的插入、删除算法的思想和实现。在进行删除一个链表不存的元素时，运行的结果并没有进入else，并打印Delete Error! 在进行检查后，发现在while循环中，使用了p->next->data != x && p->next != NULL作为条件。这个条件确保了p不会指向NULL，但它在p->next已经是NULL时仍然会尝试访问p->next->data，这是不安全的。所以将两个条件的顺序调换后代码得以正常运行并正确处理删除一个不存在元素的操作。