

//单链表基本运算算法

#include <stdio.h>

#include <malloc.h>

#include "源.h"

typedef int ElemType;

typedef struct LNode

{

ElemType data;

struct LNode\* next; //指向后继结点

} LinkNode; //声明单链表结点类型

void CreateListF(LinkNode\*& L, ElemType a[], int n)

//头插法建立单链表

{

LinkNode\* s;

L = (LinkNode\*)malloc(sizeof(LinkNode)); //创建头结点

L->next = NULL;

for (int i = 0;i < n;i++)

{

s = (LinkNode\*)malloc(sizeof(LinkNode));//创建新结点s

s->data = a[i];

s->next = L->next; //将结点s插在原开始结点之前,头结点之后

L->next = s;

}

}

void CreateListR(LinkNode\*& L, ElemType a[], int n)

//尾插法建立单链表

{

LinkNode\* s, \* r;

L = (LinkNode\*)malloc(sizeof(LinkNode)); //创建头结点

L->next = NULL;

r = L; //r始终指向终端结点,开始时指向头结点

for (int i = 0;i < n;i++)

{

s = (LinkNode\*)malloc(sizeof(LinkNode));//创建新结点s

s->data = a[i];

r->next = s; //将结点s插入结点r之后

r = s;

}

r->next = NULL; //终端结点next域置为NULL

}

void InitList(LinkNode\*& L)

{

L = (LinkNode\*)malloc(sizeof(LinkNode)); //创建头结点

L->next = NULL;

}

void DestroyList(LinkNode\*& L)//释放内存

{

LinkNode\* pre = L, \* p = pre->next;

while (p != NULL)

{

free(pre);

pre = p;

p = pre->next;

}

free(pre); //此时p为NULL,pre指向尾结点,释放它

}

bool ListEmpty(LinkNode\* L)

{

return(L->next == NULL);

}

int ListLength(LinkNode\* L)//列表长度

{

LinkNode\* p = L;int i = 0;

while (p->next != NULL)

{

i++;

p = p->next;

}

return(i);

}

void DispList(LinkNode\* L)//输出数据

{

LinkNode\* p = L->next;

while (p != NULL)

{

printf("%d ", p->data);

p = p->next;

}

printf("\n");

}

bool GetElem(LinkNode\* L, int i, ElemType& e)

{

int j = 0;

LinkNode\* p = L;

if (i <= 0) return false; //i错误返回假

while (j < i && p != NULL)

{

j++;

p = p->next;

}

if (p == NULL) //不存在第i个数据结点

return false;

else //存在第i个数据结点

{

e = p->data;

return true;

}

}

int LocateElem(LinkNode\* L, ElemType e)

{

LinkNode\* p = L->next;

int n = 1;

while (p != NULL && p->data != e)

{

p = p->next;

n++;

}

if (p == NULL)

return(0);

else

return(n);

}

bool ListInsert(LinkNode\*& L, int i, ElemType e)//插入元素

{

int j = 0;

LinkNode\* p = L, \* s;

if (i <= 0) return false; //i错误返回假

while (j < i - 1 && p != NULL) //查找第i-1个结点p

{

j++;

p = p->next;

}

if (p == NULL) //未找到位序为i-1的结点

return false;

else //找到位序为i-1的结点\*p

{

s = (LinkNode\*)malloc(sizeof(LinkNode));//创建新结点\*s

s->data = e;

s->next = p->next; //将s结点插入到结点p之后

p->next = s;

return true;

}

}

bool ListDelete(LinkNode\*& L, int i, ElemType& e)

{

int j = 0;

LinkNode\* p = L, \* q;

if (i <= 0) return false; //i错误返回假

while (j < i - 1 && p != NULL) //查找第i-1个结点

{

j++;

p = p->next;

}

if (p == NULL) //未找到位序为i-1的结点

return false;

else //找到位序为i-1的结点p

{

q = p->next; //q指向要删除的结点

if (q == NULL)

return false; //若不存在第i个结点,返回false

e = q->data;

p->next = q->next; //从单链表中删除q结点

free(q); //释放q结点

return true;

}

}

int main() {

LinkNode\* H, \* H1;

ElemType a[3] = { 1,8,4 }, i;

CreateListR(H, a, 3);

DispList(H);

ElemType e1;

ListInsert(H, 2, 2);

DispList(H);

ListDelete(H, 1, e1);

DispList(H);

ElemType length, length1;

length = ListLength(H);

printf("%d\n", length);

ElemType locate;

locate = LocateElem(H, a[1]);

printf("%d\n", locate);

ElemType\* a1;

a1 = (ElemType\*)malloc(length \* (sizeof(ElemType)));

H1 = H;

int j = 0;

while (H1->next != NULL) {

a1[j] = H1->next->data;

H1 = H1->next;

j++;

}

DestroyList(H);

CreateListF(H, a1, length);

DispList(H);

LinkNode\* L2, \* L3;

ElemType a2[5] = { 3,5,7,9,11 };

CreateListR(L2, a2, 5);

length1 = ListLength(L2);

ElemType\* a3;

a3 = (ElemType\*)malloc((length + length1) \* (sizeof(ElemType)));

int n = 0;

for (j = 1, i = 1;n < length + length1;) {

if (H->next->next != NULL) {

GetElem(H, i, a3[n]);

n++;

i++;

}

if (L2->next != NULL) {

GetElem(L2, j, a3[n]);

n++;

j++;

}

}

CreateListR(L3, a3, length + length1);

DispList(L3);

return 0;

}