#include<stdio.h>

#include<stdlib.h>

#include<stdbool.h>

#include <time.h>

#define MAX 10000

typedef int ElemType;

typedef struct LNode

{

ElemType data;

struct LNode\* next;

}LinkNode;

bool Union(LinkNode\* L1, LinkNode\* L2, LinkNode\*& L3); //并集

bool Intersection(LinkNode\* L1, LinkNode\* L2, LinkNode\*& L3); //交集

bool Difference(LinkNode\* L1, LinkNode\* L2, LinkNode\*& L3); //差集

bool InitList(LinkNode\*& L); //链表产生

bool DispList(LinkNode\* L); //链表输出

LinkNode\* Find(ElemType a, LinkNode \*L);//查找

bool Insert(ElemType a, LinkNode \*L);//插入

bool Delete(ElemType a, LinkNode \*L);//删除

int main()

{

LinkNode\* L1 = NULL, \* L2 = NULL, \* L3 = NULL, \* L4 = NULL, \* L5 = NULL;

bool flag = false;

InitList(L1);

InitList(L2);

flag = Union(L1, L2, L3);

printf("求并集：\n");

flag = DispList(L3);

flag = Intersection(L1, L2, L4);

printf("求交集：\n");

flag = DispList(L4);

flag = Difference(L1, L2, L5);

printf("求差集：\n");

flag = DispList(L5);

ElemType f;

int num;

printf("输入要查找的元素和查找的集合:num 1or2\n");

scanf("%d", &f);

if(num==1)

Find(f, L1);

else if(num==2)

Find(f, L2);

else

printf("集合错误\n");

return 0;

}

bool InitList(LinkNode\*& L) //建立单链表

{

while (!L) {

L = (LinkNode\*)malloc(sizeof(LNode));

}

L->next = NULL;

int i, n;

LinkNode\* p = NULL, \* q = NULL;

q = L;

printf("请输入数据规模：\n");

scanf("%d", &n);

srand((int)time(0));

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

while (!p) {

p = (LinkNode\*)malloc(sizeof(LNode));

}

p->data=rand();

q->next = p;

q = p;

p = q->next = NULL;

}

return true;

}

bool DispList(LinkNode\* L) //输出单链表

{

LinkNode\* p = L->next;

while (p) {

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

p = p->next;

}

printf("\n");

return true;

}

bool Union(LinkNode\* L1, LinkNode\* L2, LinkNode\*& L3) //L1、L2、L3为三个单链表的头结点

{

//如果L1和L2中均没有数据

if (L1->next == NULL && L2->next == NULL) {

return false;

}

//直到L3申请成功

while (!L3) {

L3 = (LinkNode\*)malloc(sizeof(LNode));

}

LinkNode\* p, \* q, \* r, \* s = NULL;

p = L1->next; //p为L1的工作指针

q = L2->next; //q为L2的工作指针

r = L3; //r为L3的尾指针

//当L1和L2均不为空时

while (p && q) {

s = (LinkNode\*)malloc(sizeof(LNode));

//如果申请失败继续申请，直到申请成功，因为malloc可能申请失败

while (!s) {

s = (LinkNode\*)malloc(sizeof(LNode));

}

if (p->data < q->data) {

s->data = p->data; //如果此时p指向的data的值小，复制该结点的值

r->next = s; //将s接到尾指针后面

r = r->next; //尾指针后移

p = p->next; //p继续向后扫描L1

}

else if (q->data < p->data) {

s->data =q->data; //如果此时q指向的data的值小，复制该结点的值

r->next = s; //将s接到尾指针后面

r = r->next; //尾指针后移

q = q->next; //q继续向后扫描L2

}

else {

s->data = p->data; //如果两个值相等，复制其中一个结点的值

r->next = s; //将s接到尾指针后面

r = r->next; //尾指针后移

p = p->next; //p继续向后扫描L1

q = q->next; //q继续向后扫描L2

}

}

//当p不为空时，将剩余结点接到尾指针后面

while (p) {

s = (LinkNode\*)malloc(sizeof(LNode));

//如果申请失败继续申请，直到申请成功，因为malloc可能申请失败

while (!s) {

s = (LinkNode\*)malloc(sizeof(LNode));

}

s->data = p->data; //复制结点的值

r->next = s; //将s接到尾指针后面

r = r->next; //尾指针后移

p = p->next; //p继续向后扫描L1

}

//当q不为空时，将剩余结点接到尾指针后面

while (q) {

s = (LinkNode\*)malloc(sizeof(LNode));

//如果申请失败继续申请，直到申请成功，因为malloc可能申请失败

while (!s) {

s = (LinkNode\*)malloc(sizeof(LNode));

}

s->data = q->data; //复制结点的值

r->next = s; //将s接到尾指针后面

r = r->next; //尾指针后移

q = q->next; //q继续向后扫描L2

}

r->next = NULL; //此时将尾指针后面置空，单链表L3建成

return true;

}

bool Intersection(LinkNode\* L1, LinkNode\* L2, LinkNode\*& L3) //L1、L2、L3为三个单链表的头结点

{

//如果L1和L2中均没有数据

if (L1->next == NULL && L2->next == NULL) {

return false;

}

//直到L3申请成功

while (!L3) {

L3 = (LinkNode\*)malloc(sizeof(LNode));

}

LinkNode\* p, \* q, \* r, \* s = NULL;

p = L1->next; //p为L1的工作指针

q = L2->next; //q为L2的工作指针

r = L3; //r为L3的尾指针

//当L1和L2均不为空时,只要有一个为空就不会再有交集了

while (p && q) {

s = (LinkNode\*)malloc(sizeof(LNode));

//如果申请失败继续申请，直到申请成功，因为malloc可能申请失败

while (!s) {

s = (LinkNode\*)malloc(sizeof(LNode));

}

if (p->data < q->data) {

p = p->next; //如果p偏小，继续向后扫描L1

}

else if (q->data < p->data) {

q = q->next; //如果q偏小，继续向后扫描L2

}

else {

s->data = p->data; //如果两个值相等，复制其中一个结点的值

r->next = s; //将s接到尾指针后面

r = r->next; //尾指针后移

p = p->next; //p继续向后扫描L1

q = q->next; //q继续向后扫描L2

}

}

r->next = NULL; //此时将尾指针后面置空，单链表L3建成

return true;

}

bool Difference(LinkNode\* L1, LinkNode\* L2, LinkNode\*& L3) //L1、L2、L3为三个单链表的头结点

//将L1所指链表与L2所指链表求差，将在L1中不在L2中的数据复制到L3中

{

//如果L1和L2中均没有数据

if (L1->next == NULL && L2->next == NULL) {

return false;

}

//直到L3申请成功

while (!L3) {

L3 = (LinkNode\*)malloc(sizeof(LNode));

}

LinkNode\* p, \* q, \* r, \* s = NULL;

p = L1->next; //p为L1的工作指针

q = L2->next; //q为L2的工作指针

r = L3; //r为L3的尾指针

//当L1和L2均不为空时

while (p&&q) {

s = (LinkNode\*)malloc(sizeof(LNode));

//如果申请失败继续申请，直到申请成功，因为malloc可能申请失败

while (!s) {

s = (LinkNode\*)malloc(sizeof(LNode));

}

if (p->data < q->data) {

s->data = p->data; //如果此时p指向的data的值小，复制该结点的值

r->next = s; //将s接到尾指针后面

r = r->next; //尾指针后移

p = p->next; //p继续向后扫描L1

}

else if (q->data < p->data) {

q = q->next; //q继续向后扫描L2

}

//如果两个值相等，跳过该结点

else {

p = p->next; //p继续向后扫描L1

q = q->next; //q继续向后扫描L2

}

}

//当p不为空时，将L1剩余结点接到尾指针后面

while (p) {

s = (LinkNode\*)malloc(sizeof(LNode));

//如果申请失败继续申请，直到申请成功，因为malloc可能申请失败

while (!s) {

s = (LinkNode\*)malloc(sizeof(LNode));

}

s->data = p->data; //复制结点的值

r->next = s; //将s接到尾指针后面

r = r->next; //尾指针后移

p = p->next; //p继续向后扫描L1

}

r->next = NULL; //此时将尾指针后面置空，单链表L3建成

return true;

}

LinkNode\* Find(ElemType a, LinkNode \*L){

if(!L){

printf("empty\n");

return NULL;

}

LinkNode \*p = L;

while(p){

if(p->data==a)

return p;

}

printf("can't find\n");

return NULL;

}

bool Insert(ElemType a, LinkNode \*L){

LinkNode \*p = (LinkNode \*)malloc(sizeof(LinkNode));

p->data = a;

p->next = L;

L = p;

return true;

}

bool Delete(ElemType a, LinkNode \*L){

if(!L){

printf("already delete\n");

return true;

}

LinkNode \*p = L, \*q = L;

while(p){

if(p->data==a)

break;

q = p;

p = p->next;

}

if(p){

q = p->next;

free(p);

}

printf("already delete\n");

return true;

}