#include<stdio.h>

#include<stdlib.h>

#include<Windows.h>

typedef int ElemType;

typedef struct Lnode

{

ElemType data;

struct Lnode \*next;

}Lnode,\*LinkList ;

bool CreatList(LinkList L);

void CreatList\_L(LinkList L, int n);

bool InitList\_L(LinkList &L);

bool TraverList\_L(LinkList L);

bool ListInsert\_L(LinkList &L, int i, ElemType e);

bool List\_Delete\_L(LinkList &L, int i, ElemType e);

bool UpDataList\_L(LinkList &L, int i, ElemType e);

int GetElem(LinkList L, int i, ElemType &e);

void main()

{

int m = 0;

LinkList L=NULL;

//初始化链表

InitList\_L(L);

//头插法建立链表

// CreatList\_L(L,3);

//尾插法建立链表

//CreatList(L);

//链表插入元素

ListInsert\_L(L, 1, 3);

ListInsert\_L(L, 2, 5);

ListInsert\_L(L, 3, 7);

//链表更新元素

//UpDataList\_L(L,1,9);

//删除链表元素

//List\_Delete\_L(L, 2, 5);

//获取链表中的元素

//printf("%d", GetElem(L,3,m));

//遍历链表

TraverList\_L(L);

}

//链表的初始化

bool InitList\_L(LinkList &L)

{

L = (LinkList)malloc(sizeof(Lnode));

L->next = NULL;

return true;

}

//遍历链表

bool TraverList\_L(LinkList L)

{

LinkList p = (LinkList)malloc(sizeof(Lnode));

p = L;

while (p->next!= NULL)

{

p = p->next;

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

}

return true;

}

//逆位序插入n个元素的值,建立带表头的结点的单链表L(头插法)

void CreatList\_L(LinkList L,int n)

{

for (int i = n; i > 0; --i)

{

LinkList p=(LinkList)malloc(sizeof(Lnode));

scanf\_s("%d", &p->data);

p->next = L->next;

L->next= p;

}

}

//尾插法(以999作为结束标志)

bool CreatList(LinkList L)

{

int x;

LinkList temp=(LinkList)malloc(sizeof(Lnode));

temp = L;

L->next = NULL;

temp = L;

scanf\_s("%d", &x);

while (x!=999)

{

LinkList p = (LinkList)malloc(sizeof(Lnode));

p->data = x;

temp->next = p;

temp = p;

scanf\_s("%d", &x);

}

temp->next = NULL;

return true;

}

//链表中插入元素

bool ListInsert\_L(LinkList &L, int i, ElemType e)

{

LinkList p = (LinkList)malloc(sizeof(Lnode));

p = L;

int j = 0;

while (p&&j<i-1)

{

p = p->next;

++j;

}

if (!p || j > i - 1)

return false;

LinkList s = (LinkList)malloc(sizeof(Lnode));

s->data = e;

s->next = p->next;

p->next = s;

return true;

}

//更新链表中的元素

bool UpDataList\_L(LinkList &L, int i, ElemType e)

{

LinkList p = (LinkList)malloc(sizeof(Lnode));

p = L;

int j = 0;

while (p->next&&j<i)

{

p = p->next;

++j;

}

p->data = e;

return true;

}

//删除链表的元素

bool List\_Delete\_L(LinkList &L, int i, ElemType e)

{

LinkList p = (LinkList)malloc(sizeof(Lnode));

p = L;

int j = 0;

while (p->next&&j<i - 1)

{

p = p->next;

++j;

}

if (!p->next || j > i - 1)

return false;

p->next = p->next->next;

e = p->next->data;

free(p);

return true;

}

int GetElem(LinkList L, int i, ElemType &e)

{

LinkList p = (LinkList)malloc(sizeof(Lnode));

p = L->next;

int j = 1;

while (p&&j < i)

{

p = p->next;

++j;

}

if (!p || j > i)

return false;

e = p->data;

return e;

}