**单链表的实现**

#include <stdio.h>

#include <stdlib.h>

#include <malloc.h>

//链表的Df：

//(1).由n个节点离散分配；

//(2).每个节点通过指针连接

//(3)每一个节点由一个前驱节点和一个后驱节点

//(4).首节点没有前驱节点，尾节点没有后驱节点；

typedef int ElemType;

typedef struct Node//结构名

{

ElemType data;//数据域

struct Node\* next;//指针域，指向下一节点

} LinkList;//结构名

//函数声明（若未声明，可能会有警告甚至错误）

LinkList\* initList(LinkList\* L);

LinkList\* createList(int len);

int insertLinkList(LinkList\* L, int pos, ElemType e);

int deleteLinkList(LinkList\* L, int pos, ElemType\* e);

void reverseLinkList(LinkList\* L);

int seachLinkList(LinkList\* L, ElemType e);

int getLen(LinkList\* L);

int isEmpty(LinkList\* L);

void printLinkList(LinkList\* L);

LinkList\* L;//声明

int main()

{

//target:创建元素个数，输入，显示元素，插入，显示元素，删除元素，查找元素，转置链表

ElemType e;

int len, pos;

printf("创建元素个数：");

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

printf("\n请输入：");

L = createList(len);

printf("当前链表所有元素：");

printLinkList(L);

printf("\n插入位置和插入值(中间用空格隔开)：");

scanf\_s("%d%d", &pos, &e);

insertLinkList(L, pos, e);

printf("\n插入元素后链表所有元素：");

printLinkList(L);

printf("\n请输入删除元素位置：");

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

deleteLinkList(L, pos, &e);

printf("\n元素%d已删除", e);

printf("\n删除后链表所有元素：");

printLinkList(L);

printf("\n请输入查找元素：");

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

if (seachLinkList(L, e) != -1)

{

printf("\n%d位于：%d", e, seachLinkList(L, e));

}

else printf("\n%d未找到", e);

reverseLinkList(L);

printf("\n转置后链表所有元素：");

printLinkList(L);

return 0;

}//调用函数，重点在printf

//Function part

//初始化，创建头结点

//头节点是首节点前面的那个节点，并不存放有效数据，存在的意义是方便操作

LinkList\* initList(LinkList\* L)

{

L = (LinkList\*)malloc(sizeof(LinkList));//为头结点分配空间

//malloc可以保证个数是从键盘读取的

L->next = NULL;//头结点指针域置空

//这里指向的是Node.next元素

return L;

}

//创建指定个数的单链表

LinkList\* createList(int len)

{

int i;

ElemType e;

LinkList\* L = initList(NULL),\*r,\*n;//分别定义头指针、尾指针、新指针

r = L;//尾指针初始化为头指针

for (i = 0; i < len; i++)

{

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

n = (LinkList\*)malloc(sizeof(LinkList));//申请空间

n->data = e;

n->next = NULL;//新指针指针域置空

r->next = n;//将新指针链入单链表末尾

r = r->next;//尾指针往后移

}

return L;

}

//将元素插入指定位置

int insertLinkList(LinkList\* L, int pos, ElemType e)

{

if (pos < 1 || pos > getLen(L) + 1) return 0;//插入位置错误

LinkList\* r = L, \* n;

n = (LinkList\*)malloc(sizeof(LinkList));

n->data = e;

n->next = NULL;

while (--pos > 0)

{

r = r->next;//将尾指针移动到插入位置

}

n->next = r->next;//先把新指针（插入值）链入尾指针后一个节点

r->next = n;//再把新指针（插入值）链入尾指针之后

return 1;

}

//将指定位置元素删除

int deleteLinkList(LinkList\* L, int pos, ElemType\* e)

{

if (pos < 1 || pos > getLen(L)) return 0;//删除位置错误

LinkList\* r = L, \* d;

while (--pos > 0)//自减，pos>1,实现位置确定，自减符号不出现在右边的原因是为了不和指向符号相混

{

r = r->next;//将尾指针移动到删除位置

//r->next是一个数值，即为Node.next的值

}

d = r->next;//删除元素节点

\*e = d->data;//保存删除元素值

r->next = d->next;//将尾指针跳过删除节点链入下一个节点

free(d);//释放删除节点

return 1;

}

//转置单链表：采用头插法

void reverseLinkList(LinkList\* L)

{

LinkList\* r, \* p, \* q;//定义尾指针（紧贴头指针）、欲插入指针、遍历指针

r = L->next;//尾指针紧贴头指针

p = q = r->next;//从第二个元素开始

r->next = NULL;//尾指针置空

while (q)//q相当于q != NULL

{

q = q->next;//遍历指针后移

p->next = r;//欲插入指针链入尾指针之前

L->next = p;//欲插入指针链入头指针之后

r = p;//尾指针向前移

p = q;//欲插入指针与遍历指针同步

}

}

//查找指定元素，返回指定元素位序

int seachLinkList(LinkList\* L, ElemType e)

{

if (isEmpty(L)) return -1;

int pos = 1;//位序从1开始、下标从零开始

LinkList\* r = L->next;

while (r)

{

if (r->data == e) return pos;//找到指定元素，返回位序

r = r->next;//尾指针后移

pos++;

}

return -1;//遍历完成仍未找到返回-1

}

int getLen(LinkList\* L)

{

if (L->next == NULL) return 0;//头指针指针域为空，说明单链表不含任何元素

int len = 0;

LinkList\* r = L->next;

while (r)

{

r = r->next;//尾指针后移

len++;

}

return len;

}

int isEmpty(LinkList\* L)

{

return !L->next;//L->next == NULL亦可

}

void printLinkList(LinkList\* L)

{

LinkList\* p;

p = L->next;

while (p)

{

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

p = p->next;

}

}