数据结构实验报告二

学号：201930310012 姓名：张金宇

#include<string.h>

#include<ctype.h>

#include<malloc.h> /\* malloc()等 \*/

#include<limits.h> /\* INT\_MAX等 \*/

#include<stdio.h> /\* EOF(=^Z或F6),NULL \*/

#include<stdlib.h> /\* atoi() \*/

#include<io.h> /\* eof() \*/

#include<math.h> /\* floor(),ceil(),abs() \*/

#include<process.h> /\* exit() \*/

#include <Windows.h>

#define TRUE 1

#define FALSE 0

#define OK 1

#define ERROR 0

#define INFEASIBLE -1

#pragma warning(disable:4996)

typedef int ElemType;

typedef int Status;

typedef int Boolean;

typedef struct myNode

{

ElemType data;

struct myNode\* next;

} Node;

typedef Node\* LinkList;

Status InitList(LinkList \*L)

{

\*L = (LinkList)malloc(sizeof(Node));

if (!(\*L))

exit(OVERFLOW);

(\*L)->next = NULL;

return OK;

}

Status ListLength(LinkList L)

{

LinkList p;

int i;

if (L)

{

i = 0;

p = L->next;

while (p)

{

i++;

p = p->next;

}

}

return i;

}

Status PriorElem(LinkList L, ElemType cur\_e, ElemType \*prior\_e)

{

LinkList p, p1;

if (L)

{

p = L->next;

while (p)

{

p1 = p->next;

if (p1->data == cur\_e)

{

\*prior\_e = p->data;

return OK;

}

p = p->next;

}

return ERROR;

}

else

return ERROR;

}

Status GetEle(LinkList L, int i, ElemType \*e)

{

int j;

LinkList p = L->next;

j = 1;

p = L->next;

while (p && j<i)

{

j++;

p = p->next;

}

if (!p || j>i)

return ERROR;

\*e = p->data;

return OK;

}

Status LocateElem(LinkList L, ElemType e)

{

int i = 0;

LinkList p = L->next;

while (p)

{

i++;

if (p->data == e)

p = p->next;

else

break;

}

return i;

}

Status ListInsert(LinkList L, int i, int e)

{

LinkList p, p1;

int j = 1;

p = L->next;

while (p && j<i - 1)

{

p = p->next;

++j;

}

p1 = (LinkList)malloc(sizeof(Node));

if (!p1)

exit(OVERFLOW);

p1->data = e;

p1->next = p->next;

p->next = p1;

return OK;

}

Status ListDelete(LinkList L, int i, int \*e)

{

LinkList p, p1;

int j = 1;

p = L;

while (p && j < i)

{

j++;

p = p->next;

}

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

return ERROR;

p1 = p->next;

p->next = p1->next;

\*e = p1->data;

free(p1);

return OK;

}

Status ListTraverse(LinkList L)

{

LinkList p;

p = L->next;

while (p)

{

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

p = p->next;

}

printf("\n");

return OK;

}

void CreatList(LinkList \*L, int n)

{

InitList(L);

LinkList p1;

int i;

for (i = n, p1 = \*L; i>0; i--)

{

LinkList p;

int a;

scanf("%d", &a);

//ListInsert(\*L, i, a);

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

p->data = a;

p1->next = p;

p1 = p1->next;

//\*L = p1

//p->next = (\*L)->next;

//(\*L)->next = p;

}

p1->next = NULL;

}

Status MergeList(LinkList La, LinkList \*Lb, LinkList \*Lc)

{

LinkList p1, p2, p3;

p1 = La->next;

p2 = (\*Lb)->next;

p3 = (\*Lc)->next;

while (p1 && p2)

{

if (p1->data <= p2->data)

{

p3->next = p1;

p3 = p1;

p1 = p1->next;

}

else

{

p3->next = p2;

p3 = p2;

p2 = p2->next;

}

}

p3->next = p1 ? p1 : p2;

free(\*Lb);

\*Lb = NULL;

return OK;

}

Status reservelist(LinkList L1, LinkList L2)

{

ElemType e = 0, i = 0,j=3;

ElemType \*prior\_e = 0;

LinkList a = L2;

LinkList b = a;

while (a->next)

{

a = a->next;

i++;

}

ListDelete(L2, i, &e);

while (b!=NULL)

{

a = L2;

i = 0;

while (a->next)

{

a = a->next;

i++;

}

//printf("%d ", i);

//PriorElem(L2, a->data, prior\_e);

ListInsert(L1, 5 - i + 1, a->data);

ListDelete(L2, i, &e);

b = b->next;

//j--;

}

//ListInsert(L1, 5, (L2->next)->data);

return 0;

}

/\*Status reserve(LinkList L1, LinkList L2) {

ElemType e = 0;

LinkList ptr = L2;

int i = 0;

while (ptr->next) {

ptr = ptr->next;

i++;

}

ListDelete(L2, i, &e);

//ListDelete(L2, 1, &e);

//printf("%d\n", e);

for (i = i - 1; i > 1; i--) {

ptr = L2;

while (ptr->next) {

ptr = ptr->next;

}

ListInsert(L1, 5 - i + 1, ptr->data);

ListDelete(L2, i, &e);

//printf("%d : %d\n", i, e);

}

ptr = L2;

while (ptr->next) {

ptr = ptr->next;

}

ListInsert(L1, 5 - i + 1, ptr->data);

}

\*/

int main()

{

ElemType e = 0;

LinkList L1, L2;

CreatList(&L1, 1);

CreatList(&L2, 5);

reserve(L1, L2);

//ListDelete(L2,5,&e);

ListTraverse(L2);

ListTraverse(L1);

system("pause");

return 0;

}

1.先定义两个链表L1，L2，并让a和b都为L2的头节点。其中a的作用是遍历链表，b是为了判断链表的长度。     
2.先用a遍历一次L2，然后找到最后一个元素并删除。  
3.然后重新把L2的表头赋给a，让a重新遍历L2。每次找到最后一个元素后，先把这个元素插入到L1中，然后删除。   
4.当L2只剩一个元素时，结束遍历，最后得到两个新的链表

5.一开始最后一个元素无法插入到L1中，后来发现是函数List Delete 中while的条件有错误。