实验二

#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() \*/

#define TRUE 1

#define FALSE 0

#define OK 1

#define ERROR 0

#define INFEASIBLE -1

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 ClearList(LinkList \*L) {

LinkList pre, p;

if(!(\*L))

{

return ERROR;

}

pre = (\*L)->next;

while (pre)

{

p=pre->next;

free(pre);

pre=p;

}

return OK;

}

Status DestroyList(LinkList \*L)

{

LinkList p = \*L;

while (p)

{

p=(\*L)->next;

free(\*L);

(\*L) = p;

}

return OK;

}

Status ListEmpty(LinkList L)

{

if(L!=NULL && L->next==NULL)

return OK;

else

return ERROR;

}

Status ListLength(LinkList L)

{

LinkList p;

int i;

if(L)

{

i=0;

p = L->next;

while (p)

{

i++;

p = p->next;

}

}

return i;

}

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 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 = p1->data;

return OK;

}

p = p->next;

}

return ERROR;

}

else

return ERROR;

}

Status NextElem(LinkList L, ElemType cur\_e, ElemType \*next\_e)

{

LinkList p, p1;

if(L)

{

p = L->next;

while(p && p->next)

{

p1 = p->next;

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

{

\*next\_e = p1->data;

return OK;

}

p = p->next;

}

return ERROR;

}

else

return ERROR;

}

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->next;

while(p && j<i-1)

{

j++;

p = p->next;

}

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

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;

}

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;

}

void Reverse(LinkList \*L)

{

LinkList p=(\*L)->next,q;

(\*L)->next=NULL;

while (p!=NULL)

{

q=p->next;

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

(\*L)->next=p;

p=q;

}

}

int main()

{

LinkList L;

InitList(&L);

int i=0;

for (i=1;i<10;i++)

{

ListInsert(L,i,i\*i-1);

}

printf("原线性表：\n");

ListTraverse(L);

Reverse(&L);

printf("反转后的线性表：\n");

ListTraverse(L);

return 0;

}

链表数据的逆转在于对各个结点间联系的切断与连接，通过切断两的结点间的链接，再使之与另一个结点连接实现数据的逆转，由于实验过程中对于单链表知识的不熟悉，导致代码一直报错，经过与同学的交流发现需要再定义一个单链表储存变动结点的地址，否则，将会丢失这一结点的信息，从而导致结果的错误。