题目：

1设计一个单链表，删除其中的数字字符

2两个有序单链表合并为一个有序单链表

3有三个有序线性表，删除A与BC同时相等的元素，线性表是顺序表

4从线性表中删除具有最小值的元素并由函数返回，空出的位置由最后一个元素填补，若线性表为空则显示出信息错误并退出运行

5把顺序表中的字符(字母，数字和其他字符)分成三个顺序表

6在单链表中删除节点值为奇数的节点

7在单链表中，删除重复的节点

8两个单链表合并成一个有序的单链表(元素从小到大排序)

9在单链表中将负整数放在前面，正整数放在后面

10单链表中删除元素大于min小于max 的值，单链表中的值是按递增顺序存储的

11一个顺序删除从s到t的所有元素(s小于t，s和t从键盘输入

12判断一个带头节点的单链表是否是递增的，若递增则返回1，否则返回0

**1设计一个单链表，删除其中的数字字符**

// 文件名：algo2\_2.cpp

#include<stdio.h>

#include<malloc.h>

typedef char ElemType;

typedef struct LNode

{

ElemType data;

struct LNode \*next;

} LinkList;

void InitList(LinkList \*&L)

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

L->next=NULL;

}

void DestroyList(LinkList \*&L)

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

while (q!=NULL)

{ free(p);

p=q;

q=p->next;

}

free(p);

}

bool ListEmpty(LinkList \*L)

{ return (L->next==NULL);

}

int ListLength(LinkList \*L)

{ LinkList \*p=L; int i=0;

while(p->next!=NULL)

{ i++;

p=p->next;

}

return (i);

}

void DispList(LinkList \*L)

{ LinkList \*p=L->next;

while(p!=NULL)

{ printf("%c",p-> data);

p=p->next;

}

printf("\n");

}

bool GetElem(LinkList \*L, int i, ElemType &e)

{ int j=0;

LinkList \*p=L;

while(j<i&&p!=NULL)

{ j++;

p=p->next;

}

if(p==NULL)

return false;

else

{ e=p->data;

return true;

}

}

int LocateElem(LinkList \*L, ElemType e)

{ int i=1;

LinkList \*p=L->next;

while(p!=NULL&&p->data!=e)

{ p=p->next;

i++;

}

if(p==NULL)

return (0);

else

return (i);

}

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

{ int j=0;

LinkList \*p=L,\*s;

while(j<i-1&&p!=NULL)

{ j++;

p=p->next;

}

if(p==NULL)

return false;

else

{ s=(LinkList \*)malloc(sizeof(LinkList));

s->data=e;

s->next=p->next;

p->next=s;

return true;

}

}

bool ListDelete(LinkList \*&L,int i,ElemType &e)

{ int j=0;

LinkList \*p=L,\*q;

while(j<i-1&&p!=NULL)

{ j++;

p=p->next;

}

if(p==NULL)

return false;

else

{ q=p->next;

if(q==NULL)

return false;

e=q->data;

p->next=q->next;

free(q);

return true;

}

}

//删除数字字符

void DigitDelete(LinkList \*&L){

LinkList \*p=L,\*q,\*s;

while(p!=NULL){

if((p->next->data-'0')>=0&&(p->next->data-'0')<=9){

q=p->next;

p->next=q->next;

free(q);

}

p=p->next;

}

p=L;

if((p->next->data-'0')>=0&&(p->next->data-'0')<=9){

q=p->next;

p->next=q->next;

free(q);

}

printf("\n");

}

//文件名:exp2-2.cpp

#include <stdio.h>

#include <malloc.h>

typedef char ElemType;

typedef struct LNode //定义单链表结点类型

{

ElemType data;

struct LNode \*next;

} LinkList;

extern void InitList(LinkList \*&L);

extern void DestroyList(LinkList \*&L);

extern bool ListEmpty(LinkList \*L);

extern int ListLength(LinkList \*L);

extern void DispList(LinkList \*L);

extern bool GetElem(LinkList \*L,int i,ElemType &e);

extern int LocateElem(LinkList \*L,ElemType e);

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

extern void ListDelete(LinkList \*&L);

extern void DigitDelete(LinkList \*&L);

main(){

LinkList \*h;

InitList(h);

ListInsert(h,1,'1');

ListInsert(h,2,'2');

ListInsert(h,3,'3');

ListInsert(h,4,'t');

ListInsert(h,5,'e');

ListInsert(h,6,'s');

ListInsert(h,7,'t');

ListInsert(h,8,'7');

printf("删除前：\n");

DispList(h);

DigitDelete(h);

printf("删除后：\n");

DispList(h);

}

**2两个有序单链表合并为一个有序单链表**

// 文件名：algo2\_2.cpp

#include<stdio.h>

#include<malloc.h>

#include<algorithm>

typedef char ElemType;

typedef struct LNode

{

ElemType data;

struct LNode \*next;

} LinkList;

void InitList(LinkList \*&L)

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

L->next=NULL;

}

void DestroyList(LinkList \*&L)

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

while (q!=NULL)

{ free(p);

p=q;

q=p->next;

}

free(p);

}

bool ListEmpty(LinkList \*L)

{ return (L->next==NULL);

}

int ListLength(LinkList \*L)

{ LinkList \*p=L; int i=0;

while(p->next!=NULL)

{ i++;

p=p->next;

}

return (i);

}

void DispList(LinkList \*L)

{ LinkList \*p=L->next;

while(p!=NULL)

{ printf("%c",p-> data);

p=p->next;

}

printf("\n");

}

bool GetElem(LinkList \*L, int i, ElemType &e)

{ int j=0;

LinkList \*p=L;

while(j<i&&p!=NULL)

{ j++;

p=p->next;

}

if(p==NULL)

return false;

else

{ e=p->data;

return true;

}

}

int LocateElem(LinkList \*L, ElemType e)

{ int i=1;

LinkList \*p=L->next;

while(p!=NULL&&p->data!=e)

{ p=p->next;

i++;

}

if(p==NULL)

return (0);

else

return (i);

}

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

{ int j=0;

LinkList \*p=L,\*s;

while(j<i-1&&p!=NULL)

{ j++;

p=p->next;

}

if(p==NULL)

return false;

else

{ s=(LinkList \*)malloc(sizeof(LinkList));

s->data=e;

s->next=p->next;

p->next=s;

return true;

}

}

bool ListDelete(LinkList \*&L,int i,ElemType &e)

{ int j=0;

LinkList \*p=L,\*q;

while(j<i-1&&p!=NULL)

{ j++;

p=p->next;

}

if(p==NULL)

return false;

else

{ q=p->next;

if(q==NULL)

return false;

e=q->data;

p->next=q->next;

free(q);

return true;

}

}

//比较

bool cmp(char a,char b){

return a<b;

}

//合并

void unionList(LinkList \*LA,LinkList \*LB,LinkList \*&LC)

{

int i=0;

char c[100];

LinkList \*p;

for(p=LA->next;p!=NULL;p=p->next){

c[i]=p->data;

i++;

}

for(p=LB->next;p!=NULL;p=p->next){

c[i]=p->data;

i++;

}

c[i]='/0';

/\*

for(int j=0;j<i;j++){

printf("%c",c[j]);

}

\*/

std::sort(c,c+i,cmp);

for(int j=0;j<i;j++){

ListInsert(LC,j+1,c[j]);

}

}

// 文件名：algo2\_2.cpp

#include<stdio.h>

#include<malloc.h>

#include<algorithm>

typedef char ElemType;

typedef struct LNode

{

ElemType data;

struct LNode \*next;

} LinkList;

extern void InitList(LinkList \*&L);

extern void DestroyList(LinkList \*&L);

extern bool ListEmpty(LinkList \*L);

extern int ListLength(LinkList \*L);

extern void DispList(LinkList \*L);

extern bool GetElem(LinkList \*L, int i, ElemType &e);

extern int LocateElem(LinkList \*L, ElemType e);

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

extern bool ListDelete(LinkList \*&L,int i,ElemType &e);

extern bool cmp(char a,char b);

extern void unionList(LinkList \*LA,LinkList \*LB,LinkList \*&LC);

main(){

LinkList \*h1,\*h2,\*h3;

InitList(h1);

InitList(h2);

InitList(h3);

ListInsert(h1,1,'5');

ListInsert(h1,2,'7');

ListInsert(h1,3,'1');

ListInsert(h1,4,'c');

ListInsert(h2,1,'8');

ListInsert(h2,2,'9');

ListInsert(h2,3,'2');

ListInsert(h2,4,'b');

ListInsert(h2,5,'a');

printf("合并前：\n");

DispList(h1);

DispList(h2);

printf("合并后：\n");

unionList(h1,h2,h3);

DispList(h3);

}

**3有三个有序线性表，删除A与BC同时相等的元素，线性表是顺序表**

#include<stdio.h>

#include<malloc.h>

#define MaxSize 50

typedef char ElemType;

typedef struct{

ElemType data[MaxSize];

int length;

}SqList;

void InitList(SqList \*&L){

L = (SqList \*)malloc(sizeof(SqList));

L->length = 0;

}

void DestroyList(SqList \*L){

free(L);

}

void DispList(SqList \*L){

int i;

if(L->length==0){

return;

}

for(i=0;i<L->length;i++){

printf("%c",L->data[i]);

}

printf("\n");

}

bool ListInsert(SqList \*&L,int i,ElemType e){

int j;

if(i<1||i>L->length+1){

return false;

}

i--;

for(j=L->length;j>i;j--){

L->data[j] = L->data[j-1];

}

L->data[i] = e;

L->length++;

return true;

}

bool ListDelete(SqList \*&L,int i,ElemType &e){

int j;

if(i<1||i>L->length){

return false;

}

i--;

e = L->data[i];

for(j=i;j<L->length-1;j++){

L->data[j] = L->data[j+1];

}

L->length--;

return true;

}

//删除

void Delete(SqList \*&A,SqList \*&B,SqList \*&C){

int i,j,k;

ElemType e;

for(i=0;i<A->length;i++){

for(j=0;j<B->length;j++){

for(k=0;k<C->length;k++){

if(A->data[i]==B->data[j]&&A->data[i]==C->data[k]){

ListDelete(A,i+1,e);

ListDelete(B,j+1,e);

ListDelete(C,k+1,e);

}

}

}

}

}

#include<stdio.h>

#include<malloc.h>

#define MaxSize 50

typedef char ElemType;

typedef struct{

ElemType data[MaxSize];

int length;

}SqList;

extern void InitList(SqList \*&L);

extern void DestroyList(SqList \*L);

extern void DispList(SqList \*L);

extern bool ListInsert(SqList \*&L,int i,ElemType e);

extern bool ListDelete(SqList \*&L,int i,ElemType &e);

extern void Delete(SqList \*&A,SqList \*&B,SqList \*&C);

main(){

SqList \*A,\*B,\*C;

InitList(A);

InitList(B);

InitList(C);

ListInsert(A,1,'a');

ListInsert(A,2,'b');

ListInsert(A,3,'c');

ListInsert(B,1,'a');

ListInsert(B,2,'c');

ListInsert(B,3,'d');

ListInsert(C,1,'a');

ListInsert(C,2,'b');

ListInsert(C,3,'j');

printf("删除前：\n");

DispList(A);

DispList(B);

DispList(C);

Delete(A,B,C);

printf("删除后：\n");

DispList(A);

DispList(B);

DispList(C);

}

**4从线性表中删除具有最小值的元素并由函数返回，空出的位置由最后一个元素填补，若线性表为空则显示出信息错误并退出运行**

#include<stdio.h>

#include<malloc.h>

#define MaxSize 50

typedef char ElemType;

typedef struct{

ElemType data[MaxSize];

int length;

}SqList;

void InitList(SqList \*&L){

L = (SqList \*)malloc(sizeof(SqList));

L->length = 0;

}

void DestroyList(SqList \*L){

free(L);

}

void DispList(SqList \*L){

int i;

if(L->length==0){

return;

}

for(i=0;i<L->length;i++){

printf("%c",L->data[i]);

}

printf("\n");

}

bool ListInsert(SqList \*&L,int i,ElemType e){

int j;

if(i<1||i>L->length+1){

return false;

}

i--;

for(j=L->length;j>i;j--){

L->data[j] = L->data[j-1];

}

L->data[i] = e;

L->length++;

return true;

}

bool ListDelete(SqList \*&L,int i,ElemType &e){

int j;

if(i<1||i>L->length){

return false;

}

i--;

e = L->data[i];

for(j=i;j<L->length-1;j++){

L->data[j] = L->data[j+1];

}

L->length--;

return true;

}

//删除最小元素

char Delete(SqList \*&A){

if(A->length==0){

return NULL;

}

int i;

int index=0;;

char a = A->data[0];

for(i=0;i<A->length;i++){

if(A->data[i]<a){

a = A->data[i];

index = i;

}

}

A->data[index] = A->data[A->length-1];

A->length--;

return a;

}

#include<stdio.h>

#include<malloc.h>

#define MaxSize 50

typedef char ElemType;

typedef struct{

ElemType data[MaxSize];

int length;

}SqList;

extern void InitList(SqList \*&L);

extern void DestroyList(SqList \*L);

extern void DispList(SqList \*L);

extern bool ListInsert(SqList \*&L,int i,ElemType e);

extern bool ListDelete(SqList \*&L,int i,ElemType &e);

extern char Delete(SqList \*&A);

main(){

SqList \*A;

InitList(A);

ListInsert(A,1,'5');

ListInsert(A,2,'9');

ListInsert(A,3,'4');

ListInsert(A,4,'8');

ListInsert(A,5,'2');

ListInsert(A,6,'3');

ListInsert(A,7,'a');

ListInsert(A,8,'b');

ListInsert(A,9,'c');

printf("删除前：\n");

DispList(A);

Delete(A);

printf("删除后：\n");

DispList(A);

}

**5把顺序表中的字符(字母，数字和其他字符)分成三个顺序表**

#include<stdio.h>

#include<malloc.h>

#define MaxSize 50

typedef char ElemType;

typedef struct{

ElemType data[MaxSize];

int length;

}SqList;

void InitList(SqList \*&L){

L = (SqList \*)malloc(sizeof(SqList));

L->length = 0;

}

void DestroyList(SqList \*L){

free(L);

}

void DispList(SqList \*L){

int i;

if(L->length==0){

return;

}

for(i=0;i<L->length;i++){

printf("%c",L->data[i]);

}

printf("\n");

}

bool ListInsert(SqList \*&L,int i,ElemType e){

int j;

if(i<1||i>L->length+1){

return false;

}

i--;

for(j=L->length;j>i;j--){

L->data[j] = L->data[j-1];

}

L->data[i] = e;

L->length++;

return true;

}

bool ListDelete(SqList \*&L,int i,ElemType &e){

int j;

if(i<1||i>L->length){

return false;

}

i--;

e = L->data[i];

for(j=i;j<L->length-1;j++){

L->data[j] = L->data[j+1];

}

L->length--;

return true;

}

///分表

void Separate(SqList \*&A,SqList \*&B,SqList \*&C,SqList \*&D){

int i;

int j=1;

int k=1;

int m=1;

ElemType e;

for(i=0;i<A->length;i++){

if((A->data[i]>='a'&&A->data[i]<='z')||(A->data[i]>='A'&&A->data[i]<='Z')){

//字母表

ListInsert(B,j,A->data[i]);

j++;

}else if((A->data[i]-'0')>=0&&(A->data[i]-'0')<=9){

//数字表

ListInsert(C,k,A->data[i]);

k++;

}else{

//其他

ListInsert(D,m,A->data[i]);

m++;

}

}

}

#include<stdio.h>

#include<malloc.h>

#define MaxSize 50

typedef char ElemType;

typedef struct{

ElemType data[MaxSize];

int length;

}SqList;

extern void InitList(SqList \*&L);

extern void DestroyList(SqList \*L);

extern void DispList(SqList \*L);

extern bool ListInsert(SqList \*&L,int i,ElemType e);

extern bool ListDelete(SqList \*&L,int i,ElemType &e);

extern void Separate(SqList \*&A,SqList \*&B,SqList \*&C,SqList \*&D);

main(){

SqList \*A,\*B,\*C,\*D;

InitList(A);

InitList(B);

InitList(C);

InitList(D);

ListInsert(A,1,'a');

ListInsert(A,2,'b');

ListInsert(A,3,'c');

ListInsert(A,4,'1');

ListInsert(A,5,'2');

ListInsert(A,6,'3');

ListInsert(A,7,',');

ListInsert(A,8,'.');

ListInsert(A,9,'}');

printf("分表前：\n");

DispList(A);

Separate(A,B,C,D);

printf("分表后：\n");

DispList(B);

DispList(C);

DispList(D);

}

**6在单链表中删除节点值为奇数的节点**

// 文件名：algo2\_2.cpp

#include<stdio.h>

#include<malloc.h>

typedef char ElemType;

typedef struct LNode

{

ElemType data;

struct LNode \*next;

} LinkList;

void InitList(LinkList \*&L)

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

L->next=NULL;

}

void DestroyList(LinkList \*&L)

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

while (q!=NULL)

{ free(p);

p=q;

q=p->next;

}

free(p);

}

bool ListEmpty(LinkList \*L)

{ return (L->next==NULL);

}

int ListLength(LinkList \*L)

{ LinkList \*p=L; int i=0;

while(p->next!=NULL)

{ i++;

p=p->next;

}

return (i);

}

void DispList(LinkList \*L)

{ LinkList \*p=L->next;

while(p!=NULL)

{ printf("%c",p-> data);

p=p->next;

}

printf("\n");

}

bool GetElem(LinkList \*L, int i, ElemType &e)

{ int j=0;

LinkList \*p=L;

while(j<i&&p!=NULL)

{ j++;

p=p->next;

}

if(p==NULL)

return false;

else

{ e=p->data;

return true;

}

}

int LocateElem(LinkList \*L, ElemType e)

{ int i=1;

LinkList \*p=L->next;

while(p!=NULL&&p->data!=e)

{ p=p->next;

i++;

}

if(p==NULL)

return (0);

else

return (i);

}

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

{ int j=0;

LinkList \*p=L,\*s;

while(j<i-1&&p!=NULL)

{ j++;

p=p->next;

}

if(p==NULL)

return false;

else

{ s=(LinkList \*)malloc(sizeof(LinkList));

s->data=e;

s->next=p->next;

p->next=s;

return true;

}

}

bool ListDelete(LinkList \*&L,int i,ElemType &e)

{ int j=0;

LinkList \*p=L,\*q;

while(j<i-1&&p!=NULL)

{ j++;

p=p->next;

}

if(p==NULL)

return false;

else

{ q=p->next;

if(q==NULL)

return false;

e=q->data;

p->next=q->next;

free(q);

return true;

}

}

//删除奇数节点

void Delete(LinkList \*&L){

LinkList \*p=L;

char c[100];

int i=0,k=1;

while(p->next!=NULL){

c[i]=p->next->data;

p=p->next;

i++;

}

c[i]='\0';

L->next=NULL;

for(int j=0;j<i;j++){

if(!((j+1)%2)){

ListInsert(L,k,c[j]);

k++;

}

}

}

#include<stdio.h>

#include<malloc.h>

typedef char ElemType;

typedef struct LNode

{

ElemType data;

struct LNode \*next;

} LinkList;

extern void InitList(LinkList \*&L);

extern void DestroyList(LinkList \*&L);

extern bool ListEmpty(LinkList \*L);

extern int ListLength(LinkList \*L);

extern void DispList(LinkList \*L);

extern bool GetElem(LinkList \*L, int i, ElemType &e);

extern int LocateElem(LinkList \*L, ElemType e);

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

extern bool ListDelete(LinkList \*&L,int i,ElemType &e);

extern void Delete(LinkList \*&L);

main(){

LinkList \*h;

InitList(h);

ListInsert(h,1,'1');

ListInsert(h,2,'2');//2ts7

ListInsert(h,3,'3');

ListInsert(h,4,'t');

ListInsert(h,5,'e');

ListInsert(h,6,'s');

ListInsert(h,7,'t');

ListInsert(h,8,'g');

printf("删除前：\n");

DispList(h);

Delete(h);

printf("删除后：\n");

DispList(h);

}

**7在单链表中，删除重复的节点**

// 文件名：algo2\_2.cpp

#include<stdio.h>

#include<malloc.h>

typedef char ElemType;

typedef struct LNode

{

ElemType data;

struct LNode \*next;

} LinkList;

void InitList(LinkList \*&L)

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

L->next=NULL;

}

void DestroyList(LinkList \*&L)

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

while (q!=NULL)

{ free(p);

p=q;

q=p->next;

}

free(p);

}

bool ListEmpty(LinkList \*L)

{ return (L->next==NULL);

}

int ListLength(LinkList \*L)

{ LinkList \*p=L; int i=0;

while(p->next!=NULL)

{ i++;

p=p->next;

}

return (i);

}

void DispList(LinkList \*L)

{ LinkList \*p=L->next;

while(p!=NULL)

{ printf("%c",p-> data);

p=p->next;

}

printf("\n");

}

bool GetElem(LinkList \*L, int i, ElemType &e)

{ int j=0;

LinkList \*p=L;

while(j<i&&p!=NULL)

{ j++;

p=p->next;

}

if(p==NULL)

return false;

else

{ e=p->data;

return true;

}

}

int LocateElem(LinkList \*L, ElemType e)

{ int i=1;

LinkList \*p=L->next;

while(p!=NULL&&p->data!=e)

{ p=p->next;

i++;

}

if(p==NULL)

return 0;

else

return i;

}

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

{ int j=0;

LinkList \*p=L,\*s;

while(j<i-1&&p!=NULL)

{ j++;

p=p->next;

}

if(p==NULL)

return false;

else

{ s=(LinkList \*)malloc(sizeof(LinkList));

s->data=e;

s->next=p->next;

p->next=s;

return true;

}

}

bool ListDelete(LinkList \*&L,int i,ElemType &e)

{ int j=0;

LinkList \*p=L,\*q;

while(j<i-1&&p!=NULL)

{ j++;

p=p->next;

}

if(p==NULL)

return false;

else

{ q=p->next;

if(q==NULL)

return false;

e=q->data;

p->next=q->next;

free(q);

return true;

}

}

//删除重复节点

void Delete(LinkList \*&L){

LinkList \*p=L,\*q;

InitList(q);

int k=1;

while(p->next!=NULL){

if(!LocateElem(q,p->next->data)){

//printf("%c",p->next->data);

ListInsert(q,k,p->next->data);

k++;

}

p=p->next;

}

L=q;

}

#include<stdio.h>

#include<malloc.h>

typedef char ElemType;

typedef struct LNode

{

ElemType data;

struct LNode \*next;

} LinkList;

extern void InitList(LinkList \*&L);

extern void DestroyList(LinkList \*&L);

extern bool ListEmpty(LinkList \*L);

extern int ListLength(LinkList \*L);

extern void DispList(LinkList \*L);

extern bool GetElem(LinkList \*L, int i, ElemType &e);

extern int LocateElem(LinkList \*L, ElemType e);

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

extern bool ListDelete(LinkList \*&L,int i,ElemType &e);

extern void Delete(LinkList \*&L);

main(){

LinkList \*h;

InitList(h);

ListInsert(h,1,'a');

ListInsert(h,2,'b');

ListInsert(h,3,'c');

ListInsert(h,4,'a');

ListInsert(h,5,'c');

ListInsert(h,6,'d');

ListInsert(h,7,'e');

ListInsert(h,8,'b');

ListInsert(h,9,'f');

printf("删除前：\n");

DispList(h);

Delete(h);

printf("删除后：\n");

DispList(h);

}

**8两个单链表合并成一个有序的单链表(元素从小到大排序)**

**同第2题**

**9在单链表中将负整数放在前面，正整数放在后面**

#include <stdio.h>

#include <stdlib.h>

typedef struct LNode {

int data;

struct LNode \*next;

}LinkList;

void init(LinkList \*&p,int a[],int len){

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

LinkList \*s,\*st=p;

p->next = NULL;

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

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

st->next = s;

s->data = a[i];

st = s;

}

st->next = NULL;

}

void display(LinkList \*p){

LinkList \*s=p->next;

while(s != NULL){

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

s = s->next;

}

printf("\n");

}

void destroy(LinkList \*&L){

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

while(q != NULL){

free(p);

p = q;

q = q->next;

}

}

void operate(LinkList \*&p){

LinkList \*z=p->next;

LinkList \*f=p;

LinkList \*t;

while(z->next != NULL){

if(z->next->data < 0){

t = z->next;

z->next = t->next;

t->next = f->next;

f->next = t;

f = f->next;

}else{

z = z->next;

}

}

}

#include <stdio.h>

#include <stdlib.h>

typedef struct LNode {

int data;

struct LNode \*next;

}LinkList;

extern void init(LinkList \*&p,int a[],int len);

extern void display(LinkList \*p);

extern void destroy(LinkList \*&L);

extern void operate(LinkList \*&p);

int main(){

int arr[] = {7,9,8,1,-3,6,2,-7,-5,3,2,-1};

int len = sizeof(arr)/sizeof(arr[0]);

LinkList \*L;

init(L,arr,len);

display(L);

operate(L);

display(L);

return 0;

}

**10单链表中删除元素大于min小于max 的值，单链表中的值是按递增顺序存储的**

#include <stdio.h>

#include <stdlib.h>

typedef struct LNode {

int data;

struct LNode \*next;

}LinkList;

void init(LinkList \*&p,int a[],int len){

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

LinkList \*s,\*st=p;

p->next = NULL;

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

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

st->next = s;

s->data = a[i];

st = s;

}

st->next = NULL;

}

void display(LinkList \*p){

LinkList \*s=p->next;

while(s != NULL){

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

s = s->next;

}

printf("\n");

}

void destroy(LinkList \*&L){

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

while(q != NULL){

free(p);

p = q;

q = q->next;

}

}

void operate(LinkList \*&p,int min,int max){

LinkList \*k=p,\*i,\*ind,\*t;

while(k->next != NULL){

i = k;

ind = k;

while(i->next != NULL){

if(i->next->data > min && i->next->data < max){

t = i->next;

i->next = t->next;

free(t);

}

if(ind->next->data > i->next->data){

ind = i;

}

i = i->next;

}

t = ind->next;

if(ind->next != NULL){

ind->next = ind->next->next;

}

t->next = k->next;

k->next = t;

k = k->next;

}

}

#include <stdio.h>

#include <stdlib.h>

typedef struct LNode {

int data;

struct LNode \*next;

}LinkList;

extern void init(LinkList \*&p,int a[],int len);

extern void display(LinkList \*p);

extern void destroy(LinkList \*&L);

extern void operate(LinkList \*&p,int min,int max);

int main(){

int arr[] = {7,9,8,1,-3,6,2,-7,-5,3,2,-1};

int len = sizeof(arr)/sizeof(arr[0]);

LinkList \*L;

init(L,arr,len);

display(L);

operate(L,5,9);

display(L);

return 0;}

**11一个顺序删除从s到t的所有元素(s小于t，s和t从键盘输入**

#include <stdio.h>

#include <stdlib.h>

const int MAX\_SIZE = 100;

typedef struct Node{

int data[MAX\_SIZE];

int length;

}SqList;

void init(SqList \*&L,int a[],int len){

L = (SqList \*)malloc(sizeof(SqList));

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

L->data[i] = a[i];

}

L->length = len;

}

void operate(SqList \*&L,int s,int t){

int c = t-s;

for(int i=s;i<L->length;++i){

L->data[i]=L->data[i+c];

}

L->length -= c;

}

void display(SqList \*L){

for(int i=0;i<L->length;++i){

printf("%d ", L->data[i]);

}

printf("\n");

}

#include <stdio.h>

#include <stdlib.h>

const int MAX\_SIZE = 100;

typedef struct Node{

int data[MAX\_SIZE];

int length;

}SqList;

extern void init(SqList \*&L,int a[],int len);

extern void operate(SqList \*&L,int s,int t);

extern void display(SqList \*L);

int main(){

int arr[] = {7,9,8,1,-3,6,2,-7,-5,3,2,-1};

int len = sizeof(arr)/sizeof(arr[0]);

SqList \*L;

init(L,arr,len);

int s,t;

scanf("%d%d",&s,&t);

display(L);

operate(L,s,t);

display(L);

return 0;

}

**12判断一个带头节点的单链表是否是递增的，若递增则返回1，否则返回0**

#include <stdio.h>

#include <stdlib.h>

typedef struct LNode {

int data;

struct LNode \*next;

}LinkList;

void init(LinkList \*&p,int a[],int len){

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

LinkList \*s,\*st=p;

p->next = NULL;

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

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

st->next = s;

s->data = a[i];

st = s;

}

st->next = NULL;

}

void display(LinkList \*p){

LinkList \*s=p->next;

while(s != NULL){

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

s = s->next;

}

printf("\n");

}

void destroy(LinkList \*&L){

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

while(q != NULL){

free(p);

p = q;

q = q->next;

}

}

int operate(LinkList \*p){

LinkList \*L = p->next;

while(L->next != NULL){

if(L->next->data < L->data){

return 0;

}

L = L->next;

}

return 1;

}

#include <stdio.h>

#include <stdlib.h>

typedef struct LNode {

int data;

struct LNode \*next;

}LinkList;

extern void init(LinkList \*&p,int a[],int len);

extern void display(LinkList \*p);

extern void destroy(LinkList \*&L);

extern int operate(LinkList \*p);

int main(){

int arr1[] = {1,0,3};

int len1 = sizeof(arr1)/sizeof(arr1[0]);

LinkList \*L1;

init(L1,arr1,len1);

printf("%d\n", operate(L1));

int arr2[] = {1,2,3};

int len2 = sizeof(arr2)/sizeof(arr2[0]);

LinkList \*L2;

init(L2,arr2,len2);

printf("%d\n", operate(L2));

return 0;}