**要求代码和实验报告规范，在算法思想中：对实验涉及的数据结构进行有效设计和分析；对算法进行分析并给出时间、空间复杂度的结论；清晰表达实验思路、出现的问题及解决方法。**

**一、调试成功程序及说明**

**题目：**1（1）用顺序存储结构实现书P12 ADT List 基本操作14个

**算法思想：**利用顺序表实现基本操作，结合书本，主要由size,length,data三个元素编程实现，重视空间的申请，最后由switch菜单一一运行。

**运行结果：**运行出现包含14个基本操作的菜单，初始化和创建之后便可以实现各个命令操作的要求。

**结果分析：**正确

**附源程序：**

#include<iostream>

#include<stdlib.h>

#include<malloc.h>

using namespace std;

#define TRUE 1

#define FALSE 0

#define OK 1

#define ERROR 0

#define PARA\_ERROR 0

#define OVERFLOW -2

#define LISTINITSIZE 256

#define LISTINCREMENT 128

typedef int ElemType;

typedef int Status;

typedef struct Seqlist

{

ElemType \*pData;

int length;

int size;

}Seqlist;

Status InitList(Seqlist &L)

{

L.pData=(ElemType\*) malloc (LISTINITSIZE\*sizeof(ElemType));

if(L.pData==NULL)

exit (OVERFLOW);

L.size=LISTINITSIZE;

L.length=0;

return OK;

}

Status CreateList(Seqlist &L)

{

cout<<"请输入顺序表长度：";

cin>>L.length;

cout<<"请输入顺序表元素：";

for(int i=0;i<L.length;i++)

{

cin>>L.pData[i];

}

cout<<"顺序表建立成功！"<<endl;

return OK;

}

Status DestroyList(Seqlist &L)

{

if(L.pData!=NULL)

{

free (L.pData);

L.pData=NULL;

}

L.size=0;

L.length=0;

return OK;

}

Status ClearList(Seqlist &L)

{

L.length=0;

return OK;

}

bool ListEmpty(Seqlist L)

{

if(L.length==0)

{

cout<<"是空表"<<endl;

return true;

}

else

{

cout<<"不是空表"<<endl;

return false;

}

}

Status ListLength(Seqlist L)

{

return(L.length);

}

Status GetElem(Seqlist L, int i, ElemType &e)

{

if(i<1||i>L.length)

return PARA\_ERROR;

e=L.pData[i-1];

return OK;

}

Status LocateElem(Seqlist L, ElemType e)

{

for(int i=0;i<L.length;i++)

{

if(L.pData[i]==e)

{

return i+1;

}

}

return 0;

}

Status PriorElem(Seqlist L, ElemType cur\_e, ElemType &pre\_e)

{

int location=0; //记录元素cur\_e的位置

pre\_e=-1;

location=LocateElem(L, cur\_e);

while(location==0)

{

cout<<"该元素不存在！"<<endl;

break;

}

while(location==1)

{

cout<<"该元素是第一个元素"<<endl;

break;

}

while(location!=0&&location!=1)

{

pre\_e=L.pData[location-2];

return OK;

}

}

Status NextElem(Seqlist L, ElemType cur\_e, ElemType &next\_e)

{

int location=0; //记录元素cur\_e的位置

next\_e=-1;

location=LocateElem(L, cur\_e);

while(location==0)

{

cout<<"该元素不存在！"<<endl;

break;

}

while(location==L.length)

{

cout<<"该元素是最后一个元素"<<endl;

break;

}

while(location!=0&&location!=L.length)

{

next\_e=L.pData[location];

return OK;

}

}

Status ListTraverse(Seqlist L)

{

for(int i=0; i<L.length; i++)

{

cout<<L.pData[i]<<" ";

}

cout<<endl;

return 0;

}

Status SetElem(Seqlist &L, int i, ElemType &e)

{

int t;

if(i<1||i>L.length+1)

return PARA\_ERROR;

else

{

t=L.pData[i-1];

L.pData[i-1]=e;

e=t;

}

return e;

}

Status InsertElem(Seqlist &L, int i, ElemType e)

{

if(i<1||i>L.length+1)

return PARA\_ERROR;

if(L.length>=L.size) //当前存储空间已满，需增加存储空间

{

int\*newbase=(ElemType\*)realloc(L.pData, (L.size+LISTINCREMENT)\*sizeof(ElemType));

if(newbase==NULL)

exit(OVERFLOW);

L.pData=newbase;

L.size+=LISTINCREMENT;

}

//从最后一个元素开始，直到下标为i-1的元素，依次向后挪一个位置

for(int j=L.length-1;j>=i-1;j--)

{

L.pData[j+1]=L.pData[j];

}

L.pData[i-1]=e;

L.length+=1;

return OK;

}

Status DeleteElem(Seqlist &L, int i, ElemType &e)

{

//将顺序表第i个元素删除，并用e返回

if(i<1||i>L.length)

return PARA\_ERROR;

e=L.pData[i-1];

//从第i个位置开始到最后一个元素，依次向前挪一个位置

for(int j=i; j<=L.length-1; j++)

{

L.pData[j-1]=L.pData[j];

}

L.length-=1;

return OK;

}

void menu()

{

cout<<"1.初始化 2.建立"<<endl;

cout<<"3.销毁 4.清空"<<endl;

cout<<"5.判断空表 6.元素个数"<<endl;

cout<<"7.第i个元素值 8.找相同元素位置"<<endl;

cout<<"9.前驱元素 10.后继元素"<<endl;

cout<<"11.遍历输出 12.替换元素"<<endl;

cout<<"13.插入元素 14.删除元素"<<endl;

}

int main()

{

Seqlist L;

int choice;

int e,cur\_e,pre\_e,next\_e,i=0;

while(1)

{

menu();

cout<<"请选择："<<endl;

cin>>choice;

switch(choice)

{

case(1):InitList(L);break;

case(2):CreateList(L);break;

case(3):DestroyList(L);break;

case(4):ClearList(L);break;

case(5):ListEmpty(L);break;

case(6):cout<<"顺序表长度为"<<ListLength(L)<<endl;break;

case(7):cout<<"要输出第几个元素的值？"<<endl;

cin>>i;

GetElem(L, i, e);

if(!GetElem(L, i, e))

cout<<"顺序表不存在第"<<i<<"个元素！"<<endl;

else

cout<<"第"<<i<<"个元素的值为"<<e<<endl;

break;

case(8):cout<<"请输入参数e:";

cin>>e;

if(LocateElem(L, e)==0)

{

cout<<"该元素不在顺序表内！"<<endl;

}

else if(LocateElem(L, e)!=0) cout<<"参数e的位置为"<<LocateElem(L, e)<<endl;

break;

case(9):cout<<"请输入参数e:";

cin>>cur\_e;

PriorElem(L, cur\_e, pre\_e);

while(pre\_e!=-1)

{

cout<<"e的前一个元素为"<<pre\_e<<endl;

break;

}

break;

case(10):cout<<"请输入参数e:";

cin>>cur\_e;

NextElem(L, cur\_e, next\_e);

while(next\_e!=-1)

{

cout<<"e的后一个元素为"<<next\_e<<endl;

break;

}

break;

case(11):ListTraverse(L);break;

case(12):cout<<"请输入要替换的元素位置:";

cin>>i;

cout<<"请输入新的元素值:";

cin>>e;

cout<<"替换成功！原元素值为"<<SetElem(L, i, e)<<endl;

break;

case(13):cout<<"请输入要插入的位置:";

cin>>i;

cout<<"请输入元素值:";

cin>>e;

InsertElem(L, i, e);

cout<<"插入成功！"<<endl;

break;

case(14):cout<<"请输入要删除的元素位置:";

cin>>i;

while(DeleteElem(L, i, e))

{

cout<<"删除成功！被删除的元素值为"<<e<<endl;

break;

}

break;

}

cout<<endl;

}

}

**题目：1（2）**用链式存储结构实现书P12 ADT List 基本操作14个

**算法思想：**利用链表实现基本操作，结合书本，通过链式结构的关系及指针编程实现，最后由switch菜单一一运行。

**运行结果：**运行出现包含14个基本操作的菜单，初始化和创建之后便可以实现各个命令操作的要求。

**结果分析：**正确。

**附源程序：**

#include<iostream>

#include<stdlib.h>

#include<malloc.h>

using namespace std;

#define TRUE 1

#define FALSE 0

#define OK 1

#define ERROR 0

#define PARA\_ERROR 0

#define OVERFLOW -2

typedef int ElemType;

typedef int Status;

typedef struct LNode

{

ElemType data; //数据域

struct LNode \*next; //指针域

}LNode, \*LinkList;

Status InitList(LinkList &L)

{

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

if(L==NULL) exit(OVERFLOW);

L->next=NULL;

cout<<"初始化成功！"<<endl;

return OK;

}

Status ListLength(LinkList L)

{

int n=0;

LNode \*p;

p=L->next;

while (p)

{

n++;

p=p->next;

}

return(n);

}

Status InsertElem(LinkList &L, int i, ElemType e)

{

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

if(s==NULL) exit(OVERFLOW);

s->data=e;

LNode \*p;

p=L->next;

if(i==1)

{

s->next=p;

L->next=s;

}

else if(i>1&&i<=ListLength(L))

{

for(int j=0;j<i-2;j++)

{

p=p->next;

}

s->next=p->next;

p->next=s;

}

else if(i==ListLength(L)+1)

{

for(int j=0;j<i-2;j++)

{

p=p->next;

}

p->next=s;

s->next=NULL;

}

else

{

cout<<"表中没有该位置！"<<endl;

}

return OK;

}

Status CreateList(LinkList &L)

{

int n,x;

LNode \*p,\*q;

p=L;

cout<<"请输入链表长度："<<endl;

cin>>n;

cout<<"请输入值"<<endl;

for(int i=0;i<n;i++)

{

cin>>x;

q=(LNode\*)malloc(sizeof(LNode));

if(q==NULL)

exit(OVERFLOW);

q->data =x;

p->next =q;

p=q;

p->next =NULL;

}

}

Status DestroyList(LinkList &L)

{

LNode \*p;

while(L->next!=NULL)

{

p=L->next;

L->next=p->next;

free(p); //从头结点开始逐个释放链表中的节点

}

free(L);

L=NULL;

return OK;

}

Status ClearList(LinkList &L)

{

LNode \*p,\*q;

p=L->next;

L->next=NULL; //跨过中间指向最后

while (p) //消除中间

{

q=p->next;

free(p);

p=q;

}

cout<<"清空完成!"<<endl;

}

bool ListEmpty(LinkList L)

{

if(L->next==NULL)

return TRUE;

else return FALSE;

}

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

{

if(i<1||i>ListLength(L))

return PARA\_ERROR;

LNode \*p;

p=L->next;

for(int j=1;j<i;j++)

{

p=p->next;

}

e=p->data;

return OK;

}

Status LocateElem(LinkList L, ElemType e)

{

int i=0;

LNode \*p;

p=L->next;

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

{

p=p->next;

i++;

}

if(i>=0&&i<ListLength(L))

return i+1; //返回实际位置

else if(i==ListLength(L)) return 0;

}

Status PriorElem(LinkList L, ElemType cur\_e, ElemType &pre\_e)

{

int i=LocateElem(L, cur\_e);

while(i==0)

{

cout<<"该元素不存在！"<<endl;

return ERROR;

//break;

}

while(i==1)

{

cout<<"该元素是第一个元素"<<endl;

return ERROR;

//break;

}

while(i!=0&&i!=1)

{

LNode \*p;

p=L->next;

for(int j=0;j<i-2;j++)

{

p=p->next;

}

pre\_e=p->data;

return OK;

}

}

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

{

int i=LocateElem(L, cur\_e);

if(i==0)

{

cout<<"该元素不存在！"<<endl;

return 0;

}

else if(i==ListLength(L))

{

cout<<"该元素是最后一个元素"<<endl;

return 0;

}

else

{

LNode \*p;

p=L->next;

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

{

p=p->next;

}

next\_e=p->data;

return OK;

}

}

Status ListTraverse(LinkList L)

{

LNode \*p;

p=L->next;

while(p)

{

cout<<p->data<<" ";

p=p->next;

}

cout<<endl;

return OK;

}

Status SetElem(LinkList &L, int i, ElemType &e)

{

if(i<1||i>ListLength(L))

return PARA\_ERROR;

else

{

LNode \*p;

p=L->next;

for(int j=0;j<i-1;j++)

{

p=p->next;

}

int t;

t=p->data;

p->data=e;

e=t;

}

return e;

}

Status DeleteElem(LinkList &L, int i, ElemType &e)

{

if(i<1||i>ListLength(L))

{

cout<<"表中不存在该位置！"<<endl;

return(0);

}

LNode \*p;

p=L->next;

if(i==1) //要删除第一个节点

{

e=p->data;

L->next=p->next;

free(p);

return(1);

}

else

{

for(int j=0;j<i-2;j++)

{

p=p->next;

}

LNode \*q=p->next;

e=q->data;

p->next=q->next;

free(q);

return(1);

}

}

void menu()

{

cout<<"1.初始化 2.建立"<<endl;

cout<<"3.销毁 4.清空"<<endl;

cout<<"5.判断空表 6.元素个数"<<endl;

cout<<"7.第i个元素值 8.找相同元素位置"<<endl;

cout<<"9.前驱 10.后继"<<endl;

cout<<"11.遍历输出 12.替换元素"<<endl;

cout<<"13.插入元素 14.删除元素"<<endl;

}

int main()

{

LinkList L;

int choice;

int e,cur\_e,pre\_e,next\_e,i=0;

while(1)

{

menu();

cout<<"请选择："<<endl;

cin>>choice;

switch(choice)

{

case(1):InitList(L);break;

case(2):CreateList(L);break;

case(3):DestroyList(L);break;

case(4):ClearList(L);break;

case(5):if(ListEmpty(L)) cout<<"是空表"<<endl;

else cout<<"不是空表"<<endl;

break;

case(6):cout<<"单链表长度为"<<ListLength(L)<<endl;break;

case(7):cout<<"要输出第几个元素的值？"<<endl;

cin>>i;

GetElem(L, i, e);

if(!GetElem(L, i, e))

cout<<"单链表不存在第"<<i<<"个元素！"<<endl;

else

cout<<"第"<<i<<"个元素的值为"<<e<<endl;

break;

case(8):cout<<"请输入参数e:";

cin>>e;

if(LocateElem(L, e)==0)

{

cout<<"该元素不在单链表内！"<<endl;

}

else if(LocateElem(L, e)!=0) cout<<"参数e的位置为"<<LocateElem(L, e)<<endl;

break;

case(9):cout<<"请输入参数e:";

cin>>cur\_e;

while(PriorElem(L, cur\_e, pre\_e))

{

cout<<"e的前一个元素为"<<pre\_e<<endl;

break;

}

break;

case(10):cout<<"请输入参数e:";

cin>>cur\_e;

while(NextElem(L, cur\_e, next\_e))

{

cout<<"e的后一个元素为"<<next\_e<<endl;

break;

}

break;

case(11):ListTraverse(L);break;

case(12):cout<<"请输入要替换的元素位置:";

cin>>i;

cout<<"请输入新的元素值:";

cin>>e;

cout<<"替换成功！原元素值为"<<SetElem(L, i, e)<<endl;

break;

case(13):cout<<"请输入要插入的位置:";

cin>>i;

cout<<"请输入元素值:";

cin>>e;

InsertElem(L, i, e);

cout<<"插入成功！"<<endl;

break;

case(14):cout<<"请输入要删除的元素位置:";

cin>>i;

while(DeleteElem(L, i, e))

{

cout<<"删除成功！被删除的元素值为"<<e<<endl;

break;

}

break;

}

cout<<endl;

}

}

**题目：2.**设元素值为整型的线性表L，分别采用顺序结构和链式结构存储，编写

函数，实现线性表的就地逆置（书P31 4）。

**算法思想：**就地逆置分别可由顺序表的下标和链式结构的指针实现。

**运行结果：**运行之后输入想要逆置的线性表，逆置过后的线性表会自动出现。

**结果分析：**正确

**附源程序：**

//顺序结构

#include<iostream>

#include<stdlib.h>

#include<malloc.h>

#include<math.h>

using namespace std;

#define TRUE 1

#define FALSE 0

#define OK 1

#define ERROR 0

#define PARA\_ERROR 0

#define LISTINITSIZE 256

#define LISTINCREMENT 128

typedef int ElemType;

typedef int Status;

typedef struct Seqlist

{

ElemType \*pData;

int length;

int size;

}Seqlist;

Status InitList(Seqlist &L)

{

L.pData=(ElemType\*) malloc (LISTINITSIZE\*sizeof(ElemType));

if(L.pData==NULL)

exit (OVERFLOW);

L.size=LISTINITSIZE;

L.length=0;

return OK;

}

Status CreateList(Seqlist &L)

{

cout<<"请输入顺序表长度：";

cin>>L.length;

cout<<"请输入顺序表元素：";

for(int i=0;i<L.length;i++)

{

cin>>L.pData[i];

}

return OK;

}

Status ListTraverse(Seqlist L)

{

for(int i=0; i<L.length; i++)

{

cout<<L.pData[i]<<" ";

}

cout<<endl;

return 0;

}

void ReverseList(Seqlist &L)

{

int t;

if(L.length%2==0) //元素个数为偶数

{

for(int i=0;i<L.length/2;i++)

{

t=L.pData[i];

L.pData[i]=L.pData[L.length-i-1];

L.pData[L.length-i-1]=t;

}

}

else if(L.length%2==1) //元素个数为奇数

{

for(int i=0;i<(L.length-1)/2;i++)

{

t=L.pData[i];

L.pData[i]=L.pData[L.length-i-1];

L.pData[L.length-i-1]=t;

}

}

}

int main()

{

Seqlist L;

InitList(L);

CreateList(L);

cout<<"顺序表元素为："<<endl;

ListTraverse(L);

ReverseList(L);

cout<<"逆置后的顺序表元素为："<<endl;

ListTraverse(L);

return 0;

}

//单链表

#include<iostream>

#include<stdlib.h>

#include<malloc.h>

using namespace std;

#define TRUE 1

#define FALSE 0

#define OK 1

#define ERROR 0

#define PARA\_ERROR 0

#define OVERFLOW -2

typedef int ElemType;

typedef int Status;

typedef struct LNode

{

ElemType data; //数据域

struct LNode \*next; //指针域

}LNode, \*LinkList;

Status InitList(LinkList &L)

{

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

if(L==NULL) exit(OVERFLOW);

L->next=NULL;

return OK;

}

Status CreateList(LinkList &L)

{

int n,x;

LNode \*p,\*q;

p=L;

cout<<"请输入链表长度："<<endl;

cin>>n;

cout<<"请输入值"<<endl;

for(int i=0;i<n;i++)

{

cin>>x;

q=(LNode\*)malloc(sizeof(LNode));

if(q==NULL)

exit(OVERFLOW);

q->data =x;

p->next =q;

p=q;

p->next =NULL;

}

}

Status ListTraverse(LinkList L)

{

LNode \*p;

p=L->next;

while(p)

{

cout<<p->data<<" ";

p=p->next;

}

cout<<endl;

return OK;

}

LinkList ReverseList(LinkList &L)

{

LNode \*p, \*q;

p=L->next;

L->next=NULL;

while(p)

{

q=p->next;

p->next=L->next;

L->next=p;

p=q;

}

return L;

}

int main()

{

LinkList L;

InitList(L);

CreateList(L);

cout<<"单链表原序为："<<endl;

ListTraverse(L);

ReverseList(L);

cout<<"单链表逆置后为："<<endl;

ListTraverse(L);

return 0;

}

**题目：3.**设线性表L，元素值为整型的且存在相同值，分别采用顺序结构和链式结构存储，编写函数，利用原空间，删除重复的元素值。

**算法思想：**删除重复元素分别可由顺序表的下标和链式结构的指针实现。

**运行结果：**运行之后输入想要删除重复的元素值的线性表，删除过后的线性表会自动出现。

**结果分析：**正确

**附源程序：**

//顺序结构

#include<iostream>

#include<stdlib.h>

#include<malloc.h>

using namespace std;

#define TRUE 1

#define FALSE 0

#define OK 1

#define ERROR 0

#define PARA\_ERROR 0

#define OVERFLOW -2

#define LISTINITSIZE 256

#define LISTINCREMENT 128

typedef int ElemType;

typedef int Status;

typedef struct Seqlist

{

ElemType \*pData;

int length;

int size;

}Seqlist;

Status InitList(Seqlist &L)

{

L.pData=(ElemType\*) malloc (LISTINITSIZE\*sizeof(ElemType));

if(L.pData==NULL)

exit (OVERFLOW);

L.size=LISTINITSIZE;

L.length=0;

return OK;

}

Status CreateList(Seqlist &L)

{

cout<<"请输入顺序表长度：";

cin>>L.length;

cout<<"请输入顺序表元素：";

for(int i=0;i<L.length;i++)

{

cin>>L.pData[i];

}

return OK;

}

Status ListTraverse(Seqlist L)

{

for(int i=0; i<L.length; i++)

{

cout<<L.pData[i]<<" ";

}

cout<<endl;

return 0;

}

Status DeleteRepeatList(Seqlist &L)

{

for(int i=0;i<L.length;i++)

{

for(int j=i+1;j<L.length;j++)

{

if(L.pData[j]==L.pData[i])

{

for(int k=j;k<L.length-1;k++)

{

L.pData[k]=L.pData[k+1];

}

L.length-=1;

j-=1;

}

}

}

}

int main()

{

Seqlist L;

InitList(L);

CreateList(L);

cout<<"顺序表元素为："<<endl;

ListTraverse(L);

DeleteRepeatList(L);

cout<<"删除后的顺序表元素为："<<endl;

ListTraverse(L);

}

//单链表

#include <iostream>

#include <stdlib.h>

#include <malloc.h>

using namespace std;

#define TRUE 1

#define FALSE 0

#define OK 1

#define ERROR 0

#define PARA\_ERROR 0

#define OVERFLOW -2

typedef int ElemType;

typedef int Status;

typedef struct LNode

{

ElemType data;

struct LNode \*next;

}LNode, \*LinkList;

Status InitList(LinkList &L)

{

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

if(L==NULL) exit(OVERFLOW);

L->next=NULL;

return OK;

}

Status CreateList(LinkList &L)

{

int n,x;

LNode \*p,\*q;

p=L;

cout<<"请输入链表长度："<<endl;

cin>>n;

cout<<"请输入值"<<endl;

for(int i=0;i<n;i++)

{

cin>>x;

q=(LNode\*)malloc(sizeof(LNode));

if(q==NULL)

exit(OVERFLOW);

q->data =x;

p->next =q;

p=q;

p->next =NULL;

}

}

Status ListTraverse(LinkList L)

{

LNode \*p;

p=L->next;

while(p)

{

cout<<p->data<<" ";

p=p->next;

}

cout<<endl;

return OK;

}

Status DeleteRepeatList(LinkList &L)

{

LNode \*p,\*s;

p=L->next;

while(p)

{

LNode \*q;

q=p;

while(q->next)

{

if(q->next->data==p->data)

{

s=q->next;

q->next=s->next;

free(s);

}

else q=q->next;

}

p=p->next;

}

}

int main()

{

LinkList L;

InitList(L);

CreateList(L);

cout<<"单链表元素为："<<endl;

ListTraverse(L);

DeleteRepeatList(L);

cout<<"删除后单链表元素为："<<endl;

ListTraverse(L);

}

**题目：**4.CSP题目 灰度统计直方图

**算法思想：**主要通过数组和循环的思想，本质是对数组内元素的计数。

**运行结果：**输入必要元素后，输出包含用空格分隔的若干个整数，表示输入图像的灰度直方图。

**结果分析：**正确

**附源程序：**

#include <iostream>

using namespace std;

#define N 500

#define M 500

#define L 256

int main()

{

int n,m,l,x;

int A[N][M],H[L];

cin>>n>>m>>l;

for(int i=0;i<n;i++)

{

for(int j=0;j<m;j++)

{

cin>>A[i][j];

}

}

for(int i=0;i<n;i++)

{

for(int j=0;j<m;j++)

{

x=A[i][j];

H[x]++;

}

}

for(x=0;x<l;x++)

{

cout<<H[x]<<" ";

}

}

**二、未调试成功程序及说明**

**题目：**5.CSP题目 碰撞的小球

**算法思想：**利用数组循环以及正负号的使用实现。

**错误原因：**不会编写代码对多个小球同时运动

**附源程序：**

#include <iostream>

using namespace std;

int main()

{

int L,n,t;

int flag=1;

int a,b;

for(int i=0;i<t;i++)

{

if(flag==1)

{

a++;

if(a==L||a==b)

{

flag=0;

}

}

if(flag==0)

{

a--;

if(a==0||a==b) flag=1;

}

}

}

**正确代码：**

#include<stdio.h>

#include<iostream>

using namespace std;

int main() {

int n, l, t;

int num[105];

int sort[105];

int res[105];

cin>>n>>l>>t;

int i, j, k;

for(i = 0; i < n; i++) {

cin>>num[i];

res[i] = 1;

}

for(i = 1; i <= t; i++) {

for(j = 0; j < n; j++) {

if((num[j] == l && res[j]==1) || (num[j] == 0&&res[j]==-1)) {

res[j] \*= -1;

} else {

for(k = 0; k < n; k++) {

if((num[k] == num[j]) && k != j) {

res[k] \*= -1;

res[j] \*= -1;

}

}

}

num[j] += res[j];

}

}

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

cout << num[i] << " ";

return 0;

}

1. **代码行数及小结**

行数：合计1050行左右。

小结：数据结构的初次上机，有相当一部分是书本上以及课堂上讲过的题目，需要我们提高对书本及课堂的把握度，打牢编程基础。