**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**

**1、**

**题目：**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，用选择/冒泡排序算法实现线性表的表排序。

**算法思想：**关键在于寻找最小值的选择排序，以及两两比较的冒泡排序的实现，结合上次上级代码即可。

**运行结果：**运行输入线性表，自动运算出现排序过后的线性表。

**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 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 ListTraverse(Seqlist L)

{

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

{

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

}

cout<<endl;

return 0;

}

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 ChooseSort(Seqlist &L )

{ //选择排序

int i, j, temp;

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

{

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

{

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

{

temp = L.pData[j];

L.pData[j] = L.pData[i];

L.pData[i] = temp;

}

}

}

}

Status BubbleSort(Seqlist &L )

{//冒泡排序

int i, j,t;

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

{

for( j = 0; j < L.length-i-1; j ++ )

{

if( L.pData[j] > L.pData[j+1] )

{

t=L.pData[j] ;

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

L.pData[j+1] =t;

}

}

}

}

void menu()

{

cout<<"1.选择排序法 2.冒泡排序法"<<endl;

}

int main()

{

Seqlist L;

int choice;

InitList(L);

CreateList(L);

menu();

cin>>choice;

switch(choice)

{

case(1):ChooseSort(L);

cout<<"选择排序输出线性表如下："<<endl;

ListTraverse(L);

break;

case(2):BubbleSort(L);

cout<<"冒泡排序输出线性表如下："<<endl;

ListTraverse(L);

break;

}

cout<<endl;

}

#include<iostream>

#include<stdlib.h>

#include<malloc.h>

using namespace std;

#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;

}

Status ListLength(LinkList L)

{

int n=0;

LNode \*p;

p=L->next;

while (p)

{

n++;

p=p->next;

}

return(n);

}

Status ChooseSort(LinkList &L )

{ //选择排序

LNode \*p,\*q;

p=L->next;

q=p->next ;

while(p != NULL )

{

while( q != NULL )

{

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

{

int temp = p->data ;

p->data = q->data ;

q->data =temp ;

}

q=q->next ;

}

p=p->next ;

q=p ;

}

}

Status BubbleSort( LinkList &L )

{//冒泡排序

LNode \*p,\*q;

int len=ListLength(L);

p=L;

q = L->next;

while(len>0 )

{

while( q->next != NULL )

{

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

{

int temp=q->data ;

q->data =q->next->data;

q->next->data =temp;

}

q = q->next;

}

len--;

q = p->next;

}

}

void menu()

{

cout<<"1.选择排序法 2.冒泡排序法"<<endl;

}

int main()

{

int choice;

LinkList L;

InitList(L);

CreateList(L);

menu();

cin>>choice;

switch(choice)

{

case(1):ChooseSort(L);

cout<<"选择排序输出顺序表如下："<<endl;

ListTraverse(L);

break;

case(2):BubbleSort(L);

cout<<"冒泡排序输出顺序表如下："<<endl;

ListTraverse(L);

break;

}

cout<<endl;

}

**2、**

**题目：**设线性表A、B，元素值为整型，且递减有序，编写函数，实现下列功能：对采用顺序结构和链式结构2种存储结构，要求在A的空间上构成一个新线性表C，其元素为A和B元素的并集，且表C中的元素值递减有序（互不相同）。

**算法思想：**关键在于两个线性表的连接，顺序表利用插入函数，链式结构利用指针，在利用第一题的排序并去重即可。

**运行结果：**输入两线性表AB，出现递减有序的并集C。

**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 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;

cout<<endl;

return OK;

}

Status ListTraverse(Seqlist L)

{

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

{

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

}

cout<<endl;

return 0;

}

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 BubbleSort(Seqlist &L )

{//冒泡排序

int i, j,t;

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

{

for( j = 0; j < L.length-i-1; j ++ )

{

if( L.pData[j] < L.pData[j+1] )

{

t=L.pData[j] ;

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

L.pData[j+1] =t;

}

}

}

}

Status DeleteReElem(Seqlist &L )

{ //删除重复的元素值

int l = 0;

while( l < L.length )

{

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

{

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

{

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

{

int s;

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

{

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

}

L.length -=1;

}

}

}

l ++;

}

}

Status NewList( Seqlist &La,Seqlist &Lb )

{

int lenb = Lb.length, lena = La.length, i;

for( i = lena; i < lena + lenb; i ++ ) //在La后接上Lb

{

InsertElem(La, i+1, Lb.pData[i-lena]);

}

DeleteReElem(La);

BubbleSort(La);

}

int main()

{

Seqlist La, Lb;

InitList(La);

cout<<"输入线性表A"<<endl;

CreateList(La);

InitList(Lb);

cout<<"输入线性表B"<<endl;

CreateList(Lb);

NewList(La, Lb);

cout<<"输出线性表C如下："<<endl;

ListTraverse(La);

cout<<endl;

}

//链式结构

#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

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 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 ListLength(LinkList L)

{

int n=0;

LNode \*p;

p=L->next;

while (p)

{

n++;

p=p->next;

}

return(n);

}

Status ListTraverse(LinkList L)

{

LNode \*p;

p=L->next;

while(p)

{

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

p=p->next;

}

cout<<endl;

return OK;

}

Status BubbleSort( LinkList &L )

{//冒泡排序

LNode \*p,\*q;

int len=ListLength(L);

p=L;

q = L->next;

while(len>0 )

{

while( q->next != NULL )

{

if( q->data < q->next ->data )

{

int temp=q->data ;

q->data =q->next->data;

q->next->data =temp;

}

q = q->next;

}

len--;

q = p->next;

}

}

Status DeleteReElem( LinkList &L )

{ //查找并删除重复元素

LNode \*p,\*q;

p=L->next ;

q=p->next ;

while( p != NULL )

{

while(q!= NULL)

{

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

{

p->next=q->next;

}

q=q->next ;

}

p=p->next ;

q=p;

}

}

Status NewList( LinkList &La, LinkList &Lb )

{ //连接新链表

LinkList p;

p = La->next;

while (p&&p->next != NULL)

{

p = p->next;

}

p->next = Lb->next;

BubbleSort(La);

DeleteReElem( La );

return OK;

}

int main()

{

LinkList La, Lb;

int i, n;

int &e = n;

InitList(La);

CreateList(La);

ListTraverse(La);

InitList(Lb);

CreateList(Lb);

ListTraverse(Lb);

NewList(La, Lb);

ListTraverse(La);

}

**3、**

**题目：**输入正整数n、m（m<n），设有n个人坐成一圈，从第1个人开始循环报数，报到m的人出列，然后再从下一个人开始报数，报到m的人又出列，如此重复，直到所有的人都出列为止。要求用顺序结构实现，按出列的先后顺序输出每个人的信息。

**算法思想：**顺序表做约瑟夫环，可通过标记“出列”的人实现。

**运行结果：**输入人数n及出列数m，先后输出每个人的信息。

**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 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 ListTraverse(Seqlist L)

{

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

{

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

}

cout<<endl;

return 0;

}

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 RingList( Seqlist &L, int m )

{

int i, j, k=0;

int number=0;

while(number!=L.length )

{

for( j = 0; j < L.length; j ++ )

{

if( L.pData[j] != '0' )

{

k += 1;

}

if( k == m && L.pData[j] != '0')

{

cout<<L.pData[j]<<'\n';

L.pData[j] = '0';

k = 0;

number++;

}

}

}

}

int main()

{

Seqlist L;

int i, n, m;

int s;

int &e = s;

cout<<"-进行初始化操作-"<<endl;

InitList(L); //初始化

cout<<"请输入人数n和数字m(m<n)："<<endl;

cin>>n>>m;

cout<<"请依次输入各人信息："<<endl;

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

{

cin>>s;

InsertElem(L, i, e);

}

cout<<"输出顺序表如下："<<endl;

ListTraverse(L);

cout<<endl;

cout<<"-进行约瑟夫环操作-"<<endl;

RingList(L, m);

}

**4、**

**题目：**CSP题目 非零段数组个数

**算法思想：**利用符号数flag分隔0和正数，累加计算非零段个数；再将题中所要求比较大小的p遍历比较，算出所有可能，求出最大项。

**运行结果：**输入数组后便可得到非零段个数能达到的最大值。

**结果分析：**正确

**附源程序。**

#include <iostream>

using namespace std;

#define N 1001

#define M 105

int Research(int \*p,int n)

{

int flag=1,number=0;

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

{

if(p[i]!=0)

{

if(flag==1)

{

number++;

flag=-flag;

}

}

else{

if(flag==-1)

{

flag=-flag;

}

}

}

return number;

}

int main()

{

int A[N],n,Max,max;

cin>>n;

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

{

cin>>A[i];

}

Max=Research(A,n);

for(int x=1;x<M;x++)

{

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

{

if(A[i]<x)

{

A[i]=0;

}

}

max=Research(A,n);

if(max>Max) Max=max;

}

cout<<Max;

}

**5、**

**题目：**CSP题目 相交时间段

**算法思想：**利用两个数组存储两人的时间节点，标记各个装车的时间，从0开始遍历时间，两人的时间都被标记时累加为相交的聊天时间。

**运行结果：**输入两人各装菜的时间段，输出相交的聊天总时间。

**结果分析：**正确

**附源程序。**

#include <iostream>

using namespace std;

#define N 4002

#define M 100000

int main()

{

int n,H[N],W[N];

int T[M],T1[M],sum=0;

cin>>n;

for(int i=0;i<2\*n;i++)

{

cin>>H[i];

}

for(int i=0;i<2\*n;i++)

{

cin>>W[i];

}

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

{

for(int i=0;i<2\*n;i=i+2)

{

if(t>=H[i]&&t<H[i+1])

{

T[t]=1;

}

if(t>=W[i]&&t<W[i+1])

{

T1[t]=1;

}

}

}

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

{

if(T[t]==1&&T1[t]==1)

sum++;

}

cout<<sum;

}

**二、未调试成功程序及说明**

**1、**

**题目：**输入正整数n、m（m<n），设有n个人坐成一圈，从第1个人开始循环报数，报到m的人出列，然后再从下一个人开始报数，报到m的人又出列，如此重复，直到所有的人都出列为止。要求用链式结构实现，按出列的先后顺序输出每个人的信息。

**算法思想：链式结构实现约瑟夫环，可直接删除“出列”的信息。**

**错误原因：对循环链表掌握不够。**

**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;

typedef struct CircListInfo

{

LinkList head;

LNode \*pCurNode;

int length;

}circListInfo;

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 ListTraverse(LinkList L)

{

LNode \*p;

p=L->next;

while(p)

{

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

p=p->next;

}

cout<<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 Josephus(LinkList &L, int m)

{

LNode \*pre,\*p;

pre = L;

p=L ->next;

if(p==L ) return ERROR;

while(L ->next != NULL )

{

int i=1;

while(i<m)

{

pre=p;

p=p->next;

i++;

if(p==L )

{

pre=p;

p=p->next;

}

}

cout<<p->data<<endl;

pre->next=p->next;

free(p);

p=pre->next;

if(p==L )

{

pre=p;

p=p->next;

}

}

return OK;

}

int main()

{

LinkList L;

int i, n, m;

int str;

int &e = str;

cout<<"-进行初始化操作-"<<endl;

InitList(L); //初始化

cout<<"请输入人数n和数字m(m<n)："<<endl;

cin>>n>>m;

cout<<"请依次输入各人信息："<<endl;

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

{

cin>>str;

InsertElem(L, i, e);

}

cout<<"输出链表如下："<<endl;

ListTraverse(L);

cout<<endl;

cout<<"-进行约瑟夫环操作-"<<endl;

Josephus(L, m);

}

三、代码行数及小结

行数：合计900行左右。

小结：本次上机题目实际是对上次的扩充，多处利用到了上周的基础功能函数。暴露了我对包括循环链表之内的一些知识点掌握不够，并且两道CSP题均为“暴力求解”，离更好的编程水平还有很大距离。