调试成功程序及说明

1.

题目：

算法思想：照着书上一个一个敲

运行结果：编译无错误

附源程序：

有两个程序

1.

#include<iostream>

using namespace std;

#define LIST\_INIT\_SIZE 100

#define LIST\_INCREMENT 10

#define ElemType int

typedef struct {

ElemType \*elem;

int length;

int listsize;

} SqList;

class List{

public:

void InitList(SqList &);

void DestroyList(SqList &);

void ClearList(SqList &);

int ListLength(SqList);

void GetElem(SqList,int,int &);

void LocateElem(SqList,int);

void ListInsert(SqList &,int,int);

void ListDelete(SqList &,int,ElemType &);

void TraverseList(SqList);

void Error(char \*s);

void Increment(SqList &);

};

typedef struct LNode

{

ElemType data;

struct LNode \*next;

} LNode;

typedef LNode \*LinkList;

class L\_List

{

public:

void InitList(LinkList &L);

LNode \* LocateElem(LinkList L,ElemType e);

void Error(char \*s);

void DestroyList(LinkList &L);

void ClearList(LinkList &L);

void GetElem(LinkList L,int i,ElemType &e);

void ListInsert(LinkList &L,int i,ElemType e);

void ListDelete(LinkList &L,int i,ElemType &e);

void TraverseList(LinkList L);

};

void List::Error(char \*s)

{

cout<<s<<endl;

exit(1);

}

void List::InitList(SqList &L)

{

L.elem = new ElemType[LIST\_INIT\_SIZE];

if(!L.elem) Error("Overdlow");

L.length = 0;

L.listsize = LIST\_INIT\_SIZE;

}

void List::DestroyList(SqList &L)

{

delete []L.elem;

L.length = 0;

L.listsize = 0;

}

void List::ClearList(SqList &L)

{

L.length = 0;

}

int List::ListLength(SqList L)

{

return L.length;

}

void List::GetElem(SqList L,int i,ElemType &e)

{

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

Error("Position Error");

e = L.elem[i-1];

}

int LocateElem(SqList L,ElemType e)

{

int i = 1,\*p;

p = L.elem;

while((i<=L.length)&&(\*p++!=e)) i++;

if(i<=L.length) return i;

else return 0;

}

void List::ListInsert(SqList &L,int i,ElemType e)

{

ElemType \*q,\*p;

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

{

Error(" Position Error");

}

if(L.length>=LIST\_INIT\_SIZE)

{

Increment(L);

}

q = &(L.elem[i-1]);

for(p = &(L.elem[L.length-1]);p>=q;--p)

{

\*(p+1) = \*p;

}

\*q = e;

++L.length;

}

void List::Increment(SqList &L)

{

ElemType \*newlist;

int i;

newlist = new ElemType[L.listsize+LIST\_INCREMENT];

if(! newlist) Error(" Overflow");

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

{

newlist[i] = L.elem[i];

}

delete []L.elem;

L.elem = newlist;//移交空间首地址

L.listsize += LIST\_INCREMENT;

}

void List::ListDelete(SqList &L,int i,ElemType &e)

{

ElemType \*p,\*q;

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

{

Error("Position Error");

}

e = L.elem[i-1];

p = &(L.elem[i-1]);

q = L.elem + L.length -1;

for(++p;p <= q;++p)

{

\*(p-1) = \*p;

}

--L.length;

}

void List::TraverseList(SqList L)

{

int i;

ElemType \*p;

if(L.length != 0)

{

i = 1;

p = L.elem;

while(i <= L.length)

{

cout<<\*p++;

i++;

}

}

}

//-----------------------------------------------------------------------

//-----------------------------------------------------------------------

void L\_List::InitList(LinkList &L)

{

L = new LNode;

L -> next = NULL;

}

void L\_List::Error(char \*s)

{

cout<<s<<endl;

exit(1);

}

void L\_List::DestroyList(LinkList &L)

{

LNode \*p;

while(L)

{

p = L;

L = L->next;

delete p;

}

}

void L\_List::ClearList(LinkList &L)

{

LNode \*p,\*q;

p=L->next;

while(p)

{

q = p;

p = p->next;

delete q;

}

L->next = NULL;

}

int ListLength(LinkList L)

{

LNode \*p;

p = L;

int length = 0;

while(p->next)

{

length++;

p = p->next;

}

return length;

}

void L\_List::GetElem(LinkList L,int i,ElemType &e)

{

LNode \*p;

p = L->next;

int j = 1;

while(p&&(j<i))

{

p = p->next;

++j;

}

if(!p||(j>i)) Error("Position Error");

else e = p->data;

}

LNode \* L\_List::LocateElem(LinkList L,ElemType e)

{

LNode \*p;

p = L->next;

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

p = p->next;

return p;

}

void L\_List::ListInsert(LinkList &L,int i,ElemType e)

{

LNode \*p,\*s;

p = L;

int j = 0;

while(p&&(j<i - 1))

{

p = p -> next;

++j;

}

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

{

Error("Position Error");

}

else

{

s = new LNode;

s -> data = e;

s -> next = p->next;

p -> next = s;

}

}

void L\_List::ListDelete(LinkList &L,int i,ElemType &e)

{

int j;

LNode \*p,\*q;

p = L;

j = 0;

while((p -> next)&&(j < i-1))

{

p = p->next;

++j;

}

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

{

Error("Position Error");

}

q = p -> next;

e = q -> data;

p -> next = q -> next;

delete q;

}

void L\_List::TraverseList(LinkList L)

{

LNode \*p;

p = L-> next;

while(p)

{

cout<<p->data;

p = p->next;

}

}

int main()

{

return 0;

}

2.

#include<iostream>

using namespace std;

#define ElemType int

typedef struct LNode

{

ElemType data;

struct LNode \*next;

} LNode;

typedef LNode \*LinkList;

class L\_List

{

public:

void InitList(LinkList &L);

LNode \* LocateElem(LinkList L,ElemType e);

void Error(char \*s);

void DestroyList(LinkList &L);

void ClearList(LinkList &L);

void GetElem(LinkList L,int i,ElemType &e);

void ListInsert(LinkList &L,int i,ElemType e);

void ListDelete(LinkList &L,int i,ElemType &e);

void TraverseList(LinkList L);

};

void L\_List::InitList(LinkList &L)

{

L = new LNode;

L -> next = NULL;

}

void L\_List::Error(char \*s)

{

cout<<s<<endl;

exit(1);

}

void L\_List::DestroyList(LinkList &L)

{

LNode \*p;

while(L)

{

p = L;

L = L->next;

delete p;

}

}

void L\_List::ClearList(LinkList &L)

{

LNode \*p,\*q;

p=L->next;

while(p)

{

q = p;

p = p->next;

delete q;

}

L->next = NULL;

}

int ListLength(LinkList L)

{

LNode \*p;

p = L;

int length = 0;

while(p->next)

{

length++;

p = p->next;

}

return length;

}

void L\_List::GetElem(LinkList L,int i,ElemType &e)

{

LNode \*p;

p = L->next;

int j = 1;

while(p&&(j<i))

{

p = p->next;

++j;

}

if(!p||(j>i)) Error("Position Error");

else e = p->data;

}

LNode \* L\_List::LocateElem(LinkList L,ElemType e)

{

LNode \*p;

p = L->next;

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

p = p->next;

return p;

}

void L\_List::ListInsert(LinkList &L,int i,ElemType e)

{

LNode \*p,\*s;

p = L;

int j = 0;

while(p&&(j<i - 1))

{

p = p -> next;

++j;

}

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

{

Error("Position Error");

}

else

{

s = new LNode;

s -> data = e;

s -> next = p->next;

p -> next = s;

}

}

void L\_List::ListDelete(LinkList &L,int i,ElemType &e)

{

int j;

LNode \*p,\*q;

p = L;

j = 0;

while((p -> next)&&(j < i-1))

{

p = p->next;

++j;

}

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

{

Error("Position Error");

}

q = p -> next;

e = q -> data;

p -> next = q -> next;

delete q;

}

void L\_List::TraverseList(LinkList L)

{

LNode \*p;

p = L-> next;

while(p)

{

cout<<p->data;

p = p->next;

}

}

int main()

{

return 0;

}

2.

题目：编程 建立元素值为整型的顺序表 并实现就地逆置

算法思想：

建立数组

将数组中间一位的左右元素交换

a[i]=a[i+(mid-i)\*2-1]

运行结果：

19 18 17 16….1 0

附源程序：

#include<iostream>

using namespace std;

int main()

{

int a[20],i,n,mid,temp;

n=20;

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

{

a[i]=i;

// cout<<a[i];

}

cout<<endl;

mid=n/2;

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

{

temp=a[i];

a[i]=a[i+(mid-i)\*2-1];

a[i+(mid-i)\*2-1]=temp;

cout<<i+(mid-i)\*2-1<<" ";

cout<<a[i]<<endl;

}

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

{

cout<<a[i]<<" ";

}

return 0;

}

3.

题目：编程 建立元素值为整型的单链表 并实现就地逆置

算法思想：

就地逆置 通过循环将第n个元素插入head与第一个元素之间

运行结果：

显示 19 18 17 16 15……1 0

附源程序：

#include<iostream>

using namespace std;

struct List

{

int data;

List \* next;

};

int main()

{

void init(List &L);

void rev(List &L);

void buildL(List &L);

List head;

init(head);

List \* p;

buildL(head);

p=head.next;

rev(head);

return 0;

}

void init(List &L)

{

// L =new List;

L.next = NULL;

}

void buildL(List &L)

{

int i;

List \*p,\*cur;

cur = &L;

for(i=0;i<20;i++)

{

p = new List;

p->next = NULL;

p->data = i;

cur->next = p;

cur = p;

}

}

void rev(List &L)

{

List \* p;

List \* q;

List \* s;

List \* head;

head = &L;

p=head->next;

q=p->next;

p->next = NULL;

while(q!=NULL)

{

s=q->next;

p=head->next;

head->next = q;

q->next = p;

q=s;

}

p=L.next;

while(p!=NULL)

{

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

p=p->next;

}

}

4.

题目：约瑟夫环

算法思想：

新建一个struct 存放data flag

data 默认为信息 flag 为标志 若flag 为1 则 已出局 为 0 存活

计数器 count 计算口号 1 2 3 。。。。

s=8

s=s%人数

从第s个开始计数 先循环s-1次 开始计数

t为循环的数字 该程序中默认为5

若flag=0;

count++;

若count%t==0

则出局flag=1

使用循环链表

运行结果：

显示淘汰的过程

显示存活的号码

显示剩下一个人的时候所有人的flag

附源程序：

#include<iostream>

using namespace std;

struct joseph

{

int data;

int flag;

joseph \* next;

};

int main()

{

void init(joseph &);

void linger(joseph &);

void display(joseph &);

joseph head;

init(head);

display(head);

linger(head);

display(head);

return 0;

}

void init(joseph &head)

{

joseph \* jo;

joseph \* p;

head.data=1;

head.next=NULL;

head.flag=0;

p=&head;

int i;

for(i=2;i<21;i++)

{

jo = new joseph;

jo->data=i;

jo->flag=0;

jo->next=NULL;

p->next = jo;

p=p->next;

}

p->next=&head;

}

void display(joseph &head)

{

int i=1;

joseph \*p;

p=&head;

while(i<21)

{

cout<<"p->data "<<p->data<<" p->flag "<<p->flag<<"\n";

p=p->next;

i++;

}

cout<<'\n';

}

//pretenu circle 4

// strat from s=8

void linger(joseph &head)

{

int total,count,mod,s;

joseph \*p;

p=&head;

count=0;

total=20;

mod=4;

s = 8;

s = s%total;

while(s>1)

{

p=p->next;

s--;

}

while(total>1)

{

if(!p->flag)

{

count++;

if(count%mod==0)

{

p->flag=1;

total--;

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

}

}

p=p->next;

}

while(p->flag)

{

p=p->next;

}

cout<<"\nsurvive: "<<p->data<<"\n";

}