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";

}

第二次上机实验报告

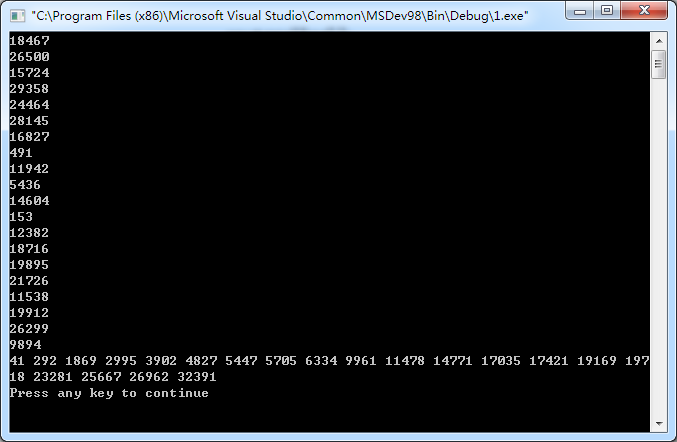
1.

题目：建立元素值为整型的顺序表，编程，用插入排序算法实现顺序表排序。

算法思想：

插入排序 从1号位置开始从前往后查询 若查到a[j]>a[i]&&j<i将a[i]插入到j前面 temp=a[i] 先将a[j]~a[n]依次往后移一位 然后赋值a[j]=temp

运行结果：



附源程序：

#include<iostream>

using namespace std;

void create(int \*a)

{

int i=0;

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

{

a[i]=rand();

cout<<rand()<<endl;

}

}

void sort(int \*a)

{

int i,j,k,temp;

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

{

for(j=0;a[i]>a[j]&&j<i;j++)

{

}

if(j<i)

{

temp=a[i];

for(k=i;k>j;k--)

{

a[k]=a[k-1];

}

a[j]=temp;

}

}

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

{

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

}

cout<<endl;

}

int main()

{

int a[20];

create(a);

sort(a);

return 0;

}

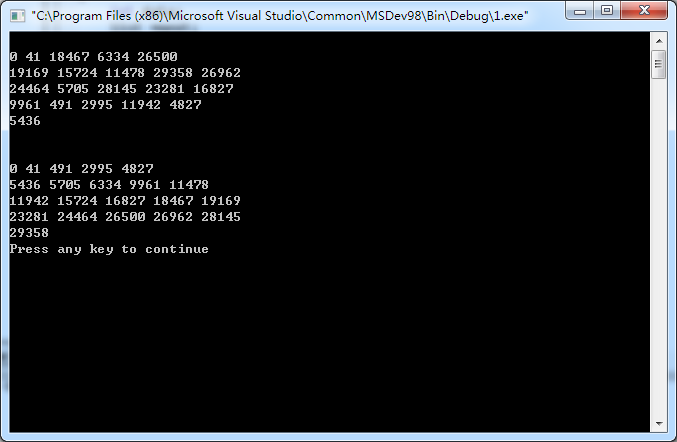
2.

题目：建立元素值为整型的单链表，编程，用插入排序算法实现单链表排序。

算法思想：

插入排序 从head开始从前往后查询 若查到p->data>q->data将p插入到q前面

运行结果：



附源程序：

#include<iostream>

using namespace std;

struct list

{

int data;

list \*next;

};

void create(list &l)

{

list \*p,\*q;

p=&l;

int i=0;

l.data=0;

l.next=NULL;

for(i=1;i<21;i++)

{

q = new list;

q->data=rand();

q->next=NULL;

p->next=q;

p=q;

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

}

}

void sort(list &l)

{

list \*p,\*q,\*s,\*r;

if(l.next==NULL)

{

exit(1);

}else{

s=l.next;

p=s->next;

if(p->data<s->data)

{

s->next=p->next;

p->next=s;

l.next=p;

}

while(p->next!=NULL)

{

r=&l;

q=r->next;

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

{

r=q;

q=q->next;

}

if(!(p==q))

{

s->next=p->next;

r->next=p;

p->next=q;

}

s=p;

p=p->next;

}

r=&l;

q=r->next;

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

{

r=q;

q=q->next;

}

if(!(p==q))

{

s->next=p->next;

r->next=p;

p->next=q;

}

cout<<endl;

}

}

void print(list &l)

{

list \*p;

p=&l;

int i=0;

while(p!=NULL)

{

if(i%5==0)

cout<<endl;

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

p=p->next;

i++;

}

cout<<endl;

}

int main()

{

list L;

create(L);

print(L);

sort(L);

print(L);

return 0;

}

3.

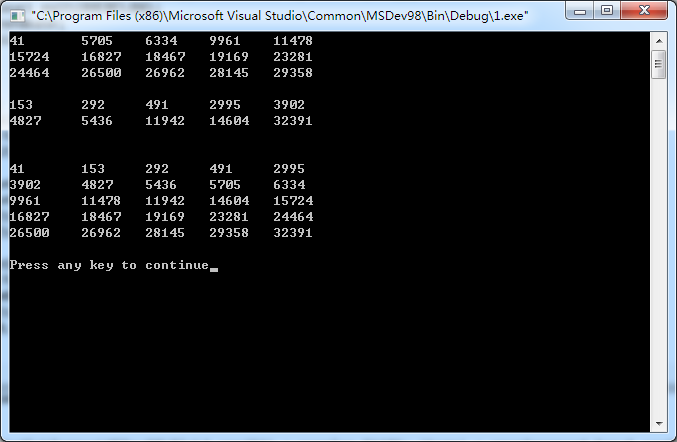
题目：

用顺序结构编程实现下列功能：假设以两个元素值为整型依值递增有序排列的线性表A和B 分别表示两个集合，现要求在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中的元素也是依值递增有序排列。

算法思想：

归并排序

运行结果：

附上源程序：

#include<iostream>

#include<stdio.h>

#define OVERFLOW -2

using namespace std;

struct list

{

int \*elem;

int length;

int listsize;

};

void create(list &l,int n)

{

int i;

l.elem=(int \*)malloc(n\*sizeof(int));

if(!l.elem) exit(OVERFLOW);

l.length=n;

l.listsize=n;

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

{

l.elem[i]=rand();

}

}

void sort(int \*a,int n)

{

int i,j,k,temp;

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

{

for(j=0;a[i]>a[j]&&j<i;j++)

{

}

if(j<i)

{

temp=a[i];

for(k=i;k>j;k--)

{

a[k]=a[k-1];

}

a[j]=temp;

}

}

// cout<<endl;

}

void print(list &l)

{

int i;

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

{

cout<<l.elem[i]<<"\t ";

if((i+1)%5==0) cout<<endl;

}

cout<<endl;

}

void m\_sort(list &l1,list &l2)

{

int i=0,j=0,k,temp;

int \*a,\*b;

a=l1.elem;

b=l2.elem;

while(i<l1.length&&j<l2.length)

{

if(a[i]>b[j])

{

for(k=l1.length;k>i;k--)

{

a[k]=a[k-1];

}

a[i]=b[j];

j++;

l1.length++;

}else{

i++;

}

}

if(i==l1.length&&i<l1.listsize)

{

for(k=i;k<l1.listsize;k++)

{

a[k]=b[j];

// cout<<k<<" "<<j<<" ";

l1.length++;

j++;

}

}

}

void recreate(list &L1,int n)

{

int \*newbase;

newbase=(int \*)realloc(L1.elem,(L1.listsize+n)\*sizeof(int));

if(!newbase) exit(OVERFLOW);

L1.elem=newbase;

L1.listsize+=n;

}

int main()

{

int n=8;

list L1,L2;

create(L1,15);

create(L2,10);

sort(L1.elem,L1.length);

sort(L2.elem,L2.length);

print(L1);

print(L2);

recreate(L1,L2.length);

m\_sort(L1,L2);

cout<<endl;

print(L1);

return 0;

}

4.

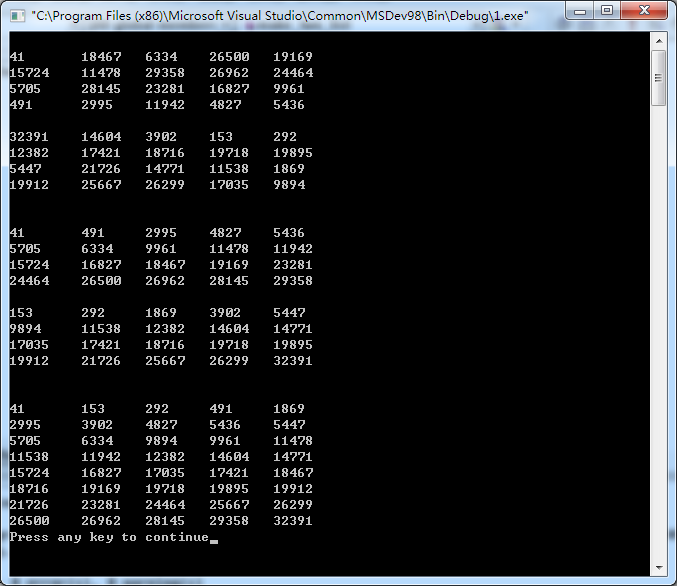
题目：

用单链表结构编程实现下列功能：假设以两个元素值为整型依值递增有序排列的线性表A和B 分别表示两个集合，现要求在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中的元素也是依值递增有序排列。用单链表结构编程实现下列功能：假设以两个元素值为整型依值递增有序排列的线性表A和B 分别表示两个集合，现要求在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中的元素也是依值递增有序排列。

算法思想：

归并排序

运行结果：



附上源程序：

#include<iostream>

using namespace std;

struct list

{

int data;

list \*next;

};

void create(list &l)

{

list \*p,\*q;

p=&l;

int i=0;

l.data=0;

l.next=NULL;

for(i=1;i<21;i++)

{

q = new list;

q->data=rand();

q->next=NULL;

p->next=q;

p=q;

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

}

}

void sort(list &l)

{

list \*p,\*q,\*s,\*r;

if(l.next==NULL)

{

exit(1);

}else{

s=l.next;

p=s->next;

if(p->data<s->data)

{

s->next=p->next;

p->next=s;

l.next=p;

}

while(p->next!=NULL)

{

r=&l;

q=r->next;

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

{

r=q;

q=q->next;

}

if(!(p==q))

{

s->next=p->next;

r->next=p;

p->next=q;

}

s=p;

p=p->next;

}

r=&l;

q=r->next;

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

{

r=q;

q=q->next;

}

if(!(p==q))

{

s->next=p->next;

r->next=p;

p->next=q;

}

cout<<endl;

}

}

void print(list &l)

{

list \*p;

p=l.next;

int i=0;

while(p!=NULL)

{

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

i++;

if(i%5==0) cout<<endl;

p=p->next;

}

}

void make\_two\_list(list &L1,list &L2)

{

create(L1);

create(L2);

cout<<endl;

print(L1);

cout<<endl;

print(L2);

sort(L1);

sort(L2);

print(L1);

cout<<endl;

print(L2);

cout<<endl;

}

void m\_sort(list &L1,list &L2,list &L3)

{

list \*p,\*q,\*m,\*r;

p=&L1;

q=L2.next;

m=&L3;

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

{

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

{

m=q->next;

q->next=p->next;

p->next=q;

q=m;

}else{

p=p->next;

}

}

if(q!=NULL)

{

p->next=q;

}

}

int main()

{

list L1,L2,L3;

L3.data=0;

L3.next=NULL;

make\_two\_list(L1,L2);

m\_sort(L2,L1,L3);

cout<<endl;

print(L1);

return 0;

}

5.

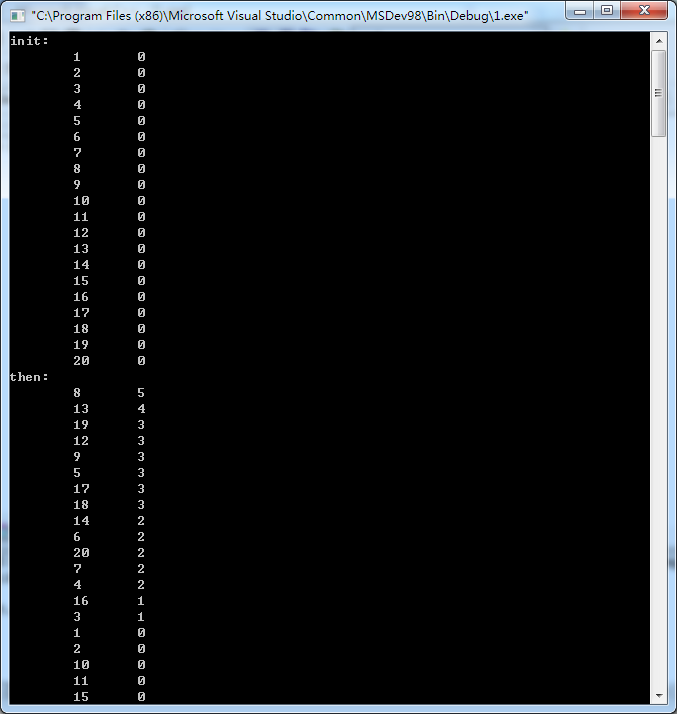
题目：

设有一个双向循环链表，每个结点中除有pre，data和next三个域外，还增设了一个访问频度域freq。在链表被起作用前，频度域freq的值均初始化为零，而当对链表进行一次LOCATE(L,x)的操作之后，被访问的结点（即元素值等于x的节点）中的频度域freq的值便增1，同时调整链表中结点之间的次序，使其按访问频度非递增的顺序排列，以便始终保持被频繁访问的结点总是靠近表头结点。试编程实现之。

算法思想：

插入排序

运行结果：



附上源程序：

#include<iostream>

#include<stdio.h>

#define OVERFLOW -2

using namespace std;

struct both{

int data;

both \* pre;

both \* next;

int freq;

};

void create(both &l)

{

int i;

both \*p,\*q;

p=&l;

l.data=0;

l.freq=0;

l.next=NULL;

for(i=1;i<21;i++)

{

q = new both;

q->data=i;

q->next=NULL;

q->pre=NULL;

q->freq=0;

p->next=q;

q->pre=p;

p=q;

}

}

void sort(both \*p,both &L)

{

both \*s,\*q,\*r;

s=p->pre;

while(((p->freq)>(s->freq))&&s!=&L)

{

s=s->pre;

}

if(p->next!=NULL)

{

p->pre->next=p->next;

p->next->pre=p->pre;

p->pre=s;

p->next=s->next;

s->next->pre=p;

s->next=p;

}else{

p->pre->next=p->next;

p->pre=s;

p->next=s->next;

s->next->pre=p;

s->next=p;

}

}

void Locate(both &L,int x)

{

int flag=0;

both \*p,\*q;

p=L.next;

while(p!=NULL)

{

if(p->data==x)

{

p->freq++;

sort(p,L);

flag=1;

}

p=p->next;

}

}

void print(both &L)

{

both \*p;

p=L.next;

while(p!=NULL)

{

cout<<"\t"<<p->data<<"\t"<<p->freq<<endl;

p=p->next;

}

}

int main()

{

both head;

int i,t;

create(head);

cout<<"init:"<<endl;

print(head);

int n=rand()%200;

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

{

t=rand()%21;

Locate(head,t);

}

cout<<"then:"<<endl;

print(head);

return 0;

}

第三次上机实验报告

1.

题目：编程实现书P59 ADT Stacke基本操作9个，用顺序存储结构实现；

算法思想：

照着书上一个一个敲

运行结果:

编译无bug

附源代码：

#include<iostream>

using namespace std;

typedef int Status;

#define TRUE 1

#define FALSE 0

#define OK 1

#define ERROR 0

#define INFEASIBLE -1

#define OVERFLOW -2

typedef int ElemType;

typedef struct{

ElemType \* base;

ElemType top;

int stacksize;

}SqStack;

#define STACK\_INIT\_SIZE 100

#define STACKINCREMENT 10

Status InitStack(SqStack &S);

Status DestroyStack(SqStack &S);

Status ClearStack(SqStack &S);

int StackLength(SqStack &S);

Status GetTop(SqStack &S,ElemType &e);

Status Push(SqStack &S,ElemType e);

Status Pop(SqStack &S,ElemType &e);

Status StackTraverse(SqStack &S,Status (\*visit)());

void Increment(SqStack &S);

//从栈底到栈顶依次对栈中的每个元素调用visit()

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

Status InitStack(SqStack &S)

{

S.base = (ElemType \*)malloc(STACK\_INIT\_SIZE \* sizeof(ElemType));

if(!S.base) exit(OVERFLOW);

S.top = -1;

S.stacksize = STACK\_INIT\_SIZE;

return OK;

}

Status DestroyStack(SqStack &S){

delete []S.base;

S.top=-1;

S.stacksize=0;

return OK;

}

Status ClearStack(SqStack &S){

S.top=-1;

return OK;

}

int StackLength(SqStack &S){

return (S.top+1);

}

Status GetTop(SqStack &S,ElemType &e){

if(S.top == -1) return ERROR;

e = S.base[S.top];

return OK;

}

Status Push(SqStack &S,ElemType e){

if(S.top>=S.stacksize){

S.base=(ElemType\*)realloc(S.base,(S.stacksize+STACKINCREMENT)\*sizeof(ElemType));

if(!S.base) exit (OVERFLOW);

// S.top = S.base + S.stacksize;

S.stacksize += STACKINCREMENT;

}

S.base[S.top++] =e;

return OK;

}

Status Pop(SqStack &S,ElemType &e){

if(S.top==-1) return ERROR;

e = S.base[S.top--];

return OK;

}

// Status StackTraverse(SqStack &S,Status (\*visit)()){

// }

void Increment(SqStack &S)

{

int i;

ElemType \*newstack;

newstack = new ElemType[S.stacksize+STACKINCREMENT];

if(!newstack) ERROR("OVERFLOW");

for(i=0;i<S.top;i++)

{

newstack[i]=S.base[i];

}

delete []S.base;

S.base=newstack;

S.stacksize+=STACKINCREMENT;

}

int main()

{

return 0;

}

2.

题目：编程实现书P59 ADT Queue基本操作9个，用链式存储结构实现；

算法思想：

照着书上一个一个敲

运行结果:

编译无bug

附源代码：

#include<iostream>

using namespace std;

#define QUEUE\_MAX\_SIZE 100

typedef int ElemType;

typedef struct QNode{

ElemType data;

struct QNode \*next;

}QNode,\*QueuePtr;

typedef struct{

QueuePtr front;

QueuePtr rear;

} LinkQueue;

void InitQueue\_L(LinkQueue &Q);

void DestroyQueue\_L(LinkQueue &Q);

void ClearQueue\_L(LinkQueue &Q);

int QueueLength\_L(LinkQueue Q);

void GetHead\_Sq\_L(LinkQueue Q,ElemType &e);

void EnQueue\_L(LinkQueue &Q,ElemType e);

void DeQueue\_L(LinkQueue &Q,ElemType &e);

void ERROR(char \*s)

{

cout<<s<<endl;

exit(1);

}

void InitQueue\_L(LinkQueue &Q)

{

Q.front=Q.rear=new QNode;

Q.front->next=NULL;

}

void DestroyQueue\_L(LinkQueue &Q)

{

while(Q.front)

{

Q.rear=Q.front->next;

delete Q.front;

Q.front=Q.rear;

}

}

void ClearQueue\_L(LinkQueue &Q)

{

QNode \*p,\*q;

p=Q.front->next;

while(p)

{

q=p;

p=p->next;

delete q;

}

Q.front->next=NULL;

Q.rear=Q.front;

}

int QueueLength\_L(LinkQueue Q)

{

QNode \*p;

int length=0;

p=Q.front;

while(p->next)

{

length++;

p=p->next;

}

return length;

}

void GetHead\_Sq\_L(LinkQueue Q,ElemType &e)

{

if(Q.front->next==NULL) ERROR("Queue Empty");

e=Q.front->next->data;

}

void EnQueue\_L(LinkQueue &Q,ElemType e)

{

QNode \*p;

p=new QNode;

p->data=e;

p->next=NULL;

Q.rear->next=p;

}

void DeQueue\_L(LinkQueue &Q,ElemType &e)

{

QNode \*p;

if(Q.front->next==NULL)

{

ERROR("Queue Empty");

}

p=Q.front->next;

e=p->data;

Q.front->next=p->next;

if(Q.rear==p) Q.rear=Q.front;

delete p;

}

int main()

{

return 0;

}

3.

题目：试利用循环队列编写求k阶斐波那契序列中前n+1项的算法。

要求满足：

,其中max为某个约定常数。(注意：本题所用循环队列的容量仅为k,则在算法执行结束时，留在循环队 列中的元素应是所求k阶斐波那契序列中的最后k项)

算法思想：

K阶斐波那契数列

定义：第k和k+1项为1，前k - 1项为0，从k项之后每一项都是前k项的和

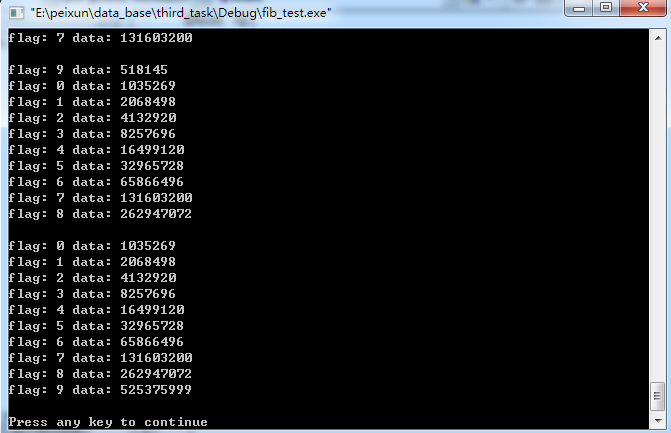
化简一下，得到迭代公式：

①：f(m)=f(m-1)+f(m-2)+…+f(m-k)

②：f(m-1)=f(m-2)+f(m-3)+…+f(m-k-1)

①-②: f(m)-f(m-1)=f(m-1)-f(m-k-1)

f(m)=2f(m-1)-f(m-k-1)

运行结果：

每一次的移动都会显示出来

通过添加flag方便调试

附源程序：

#include<iostream>

using namespace std;

#define QUEUE\_MAX\_SIZE 100

#define MAX\_NUM 400000000

#define k 10

typedef int ElemType;

typedef struct QNode{

ElemType data;

int flag;

struct QNode \*next;

}QNode,\*QueuePtr;

typedef struct{

QueuePtr front;

QueuePtr rear;

} LinkQueue;

void InitQueue\_L(LinkQueue &Q);

void DestroyQueue\_L(LinkQueue &Q);

void ClearQueue\_L(LinkQueue &Q);

int QueueLength\_L(LinkQueue Q);

void GetHead\_Sq\_L(LinkQueue &Q,ElemType &e);

void EnQueue\_L(LinkQueue &Q,ElemType e);

void DeQueue\_L(LinkQueue &Q,ElemType &e);

void ERROR(char \*s)

{

cout<<s<<endl;

exit(1);

}

void InitQueue\_L(LinkQueue &Q)

{

Q.front=Q.rear=new QNode;

Q.front->next=NULL;

Q.front->flag=0;

Q.front->data=0;

Q.rear->next=Q.front;

}

void DestroyQueue\_L(LinkQueue &Q)

{

while(Q.front)

{

Q.rear=Q.front->next;

delete Q.front;

Q.front=Q.rear;

}

}

void ClearQueue\_L(LinkQueue &Q)

{

QNode \*p,\*q;

p=Q.front->next;

while(p)

{

q=p;

p=p->next;

delete q;

}

Q.front->next=NULL;

Q.rear=Q.front;

}

int QueueLength\_L(LinkQueue Q)

{

QNode \*p;

int length=0;

p=Q.front;

while(p->next)

{

length++;

p=p->next;

}

return length;

}

void GetHead\_Sq\_L(LinkQueue &Q,ElemType &e)

{

if(Q.front->next==NULL) ERROR("Queue Empty");

e=Q.front->data;

Q.front=Q.front->next;

}

void EnQueue\_L(LinkQueue &Q,ElemType e,int i)

{

QNode \*p;

p=new QNode;

p->data=e;

p->flag=i+1;

p->next=NULL;

Q.rear->next=p;

Q.rear=p;

}

void DeQueue\_L(LinkQueue &Q,ElemType &e)

{

QNode \*p;

if(Q.front->next==NULL)

{

ERROR("Queue Empty");

}

p=Q.front->next;

e=p->data;

Q.front->next=p->next;

if(Q.rear==p) Q.rear=Q.front;

delete p;

}

void update\_Queue(LinkQueue &Q,ElemType e)

{

Q.rear=Q.rear->next;

Q.rear->data=e;

}

void print(LinkQueue &Q)

{

QNode \*p;

p=Q.front;

cout<<"flag: "<<p->flag<<" data: "<<p->data<<endl;

p=p->next;

while(p!=Q.front)

{

cout<<"flag: "<<p->flag<<" data: "<<p->data<<endl;

p=p->next;

}

cout<<endl;

}

int main()

{

LinkQueue Q;

int e;

int i,temp;

InitQueue\_L(Q);

for(i=0;i<k-2;i++)

{

EnQueue\_L(Q,0,i);

cout<<endl<<i;

}

EnQueue\_L(Q,1,k-2);

Q.rear->next=Q.front;

GetHead\_Sq\_L(Q,e);

update\_Queue(Q,1);

temp=1;

while(temp<MAX\_NUM)

{

GetHead\_Sq\_L(Q,e);

update\_Queue(Q,temp\*2-e);

temp=temp\*2-e;

print(Q);

}

return 0;

}