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

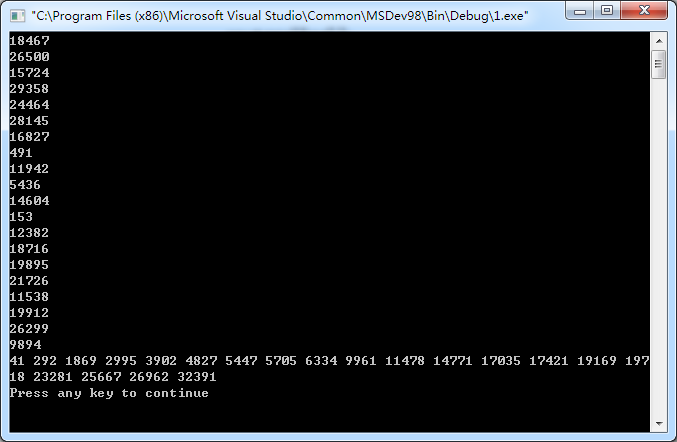
1.

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;

}

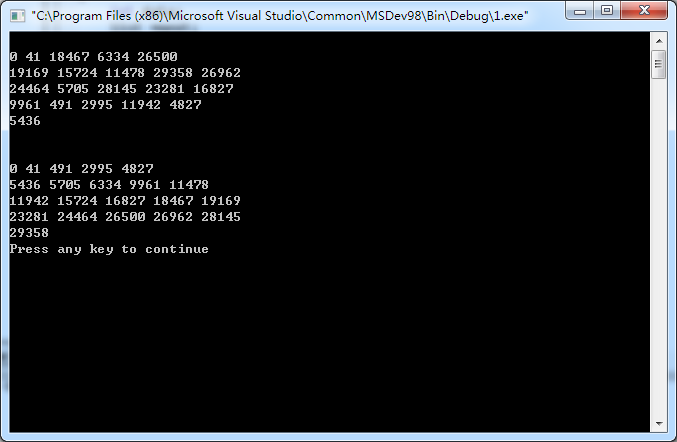
2.

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.

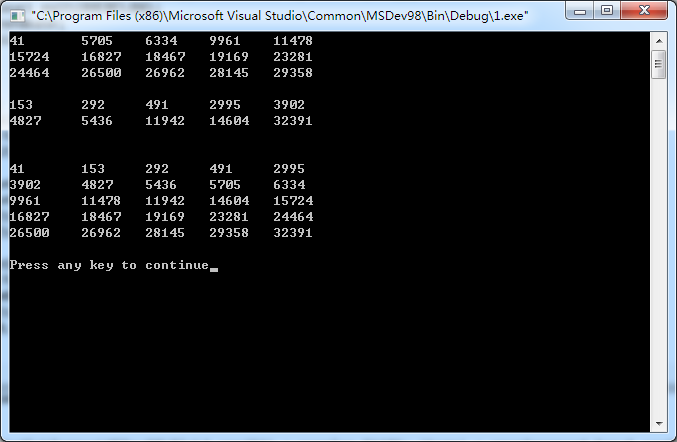
题目：

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.

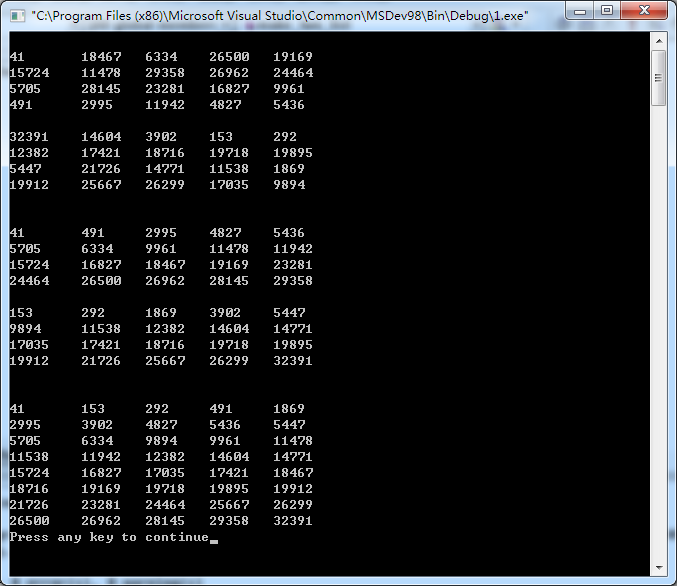
题目：

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.

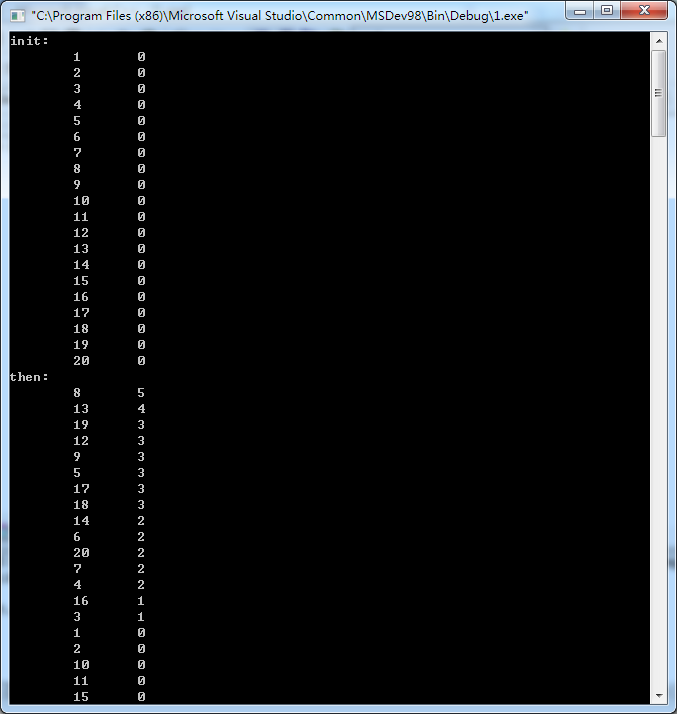
题目：

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

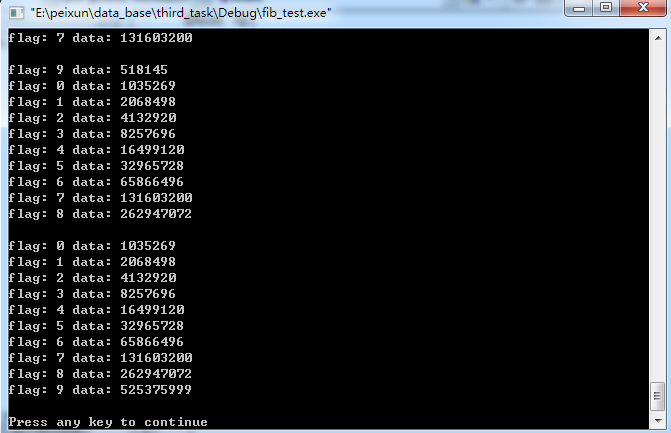
化简一下，得到迭代公式：

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;

}

第四次上机实验报告

1.

题目：输入稀疏矩阵，建立稀疏矩阵三元组顺序结构，实现转置（1、2）；

算法思想：

装置前的矩阵每一列不为零的数的数目==装置后每一行的不为零的书的数目

故新建辅助数组

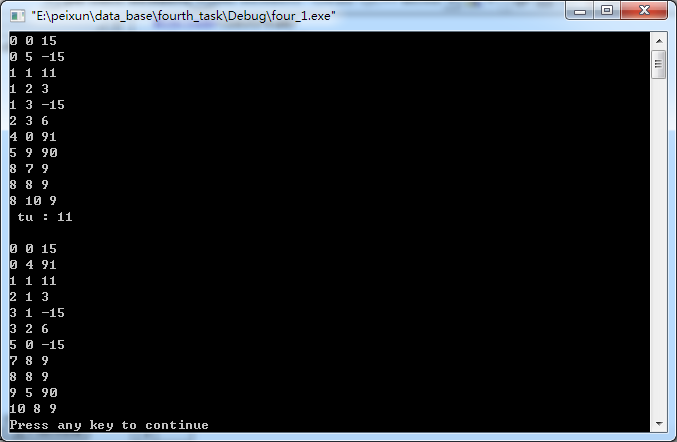
num[n]:存储每一列非零个数 cpot[n]：存储装置后矩阵每一行开始的下标

cpot[0]=1;

cpot[n]=cpot[n-1]+num[n-1]

遍历一遍原矩阵存储 便可实现装置

运行结果：



附源程序：

#include<iostream>

using namespace std;

#include<fstream>

#define TriList\_Size 1000

typedef int Element;

typedef struct {

int row,col;

Element e;

}Triple;

typedef struct {

Triple data[TriList\_Size+1];

int mu,nu,tu;

}TSMatrix;

void InitSMatrix(TSMatrix &M)

{

M.tu=0;

M.nu=0;

M.mu=0;

}

void DestroySMatrix(TSMatrix &M);

void ReadSMatrix(TSMatrix &M)

{

fstream f1;

int i=0;

f1.open("test.txt",ios::in);

f1>>M.mu>>M.nu;

while(!f1.eof())

{

f1>>M.data[i].row>>M.data[i].col>>M.data[i].e;

M.tu++;

i++;

}

for(i=0;i<M.tu;i++)

{

cout<<M.data[i].row<<" "<<M.data[i].col<<" "<<M.data[i].e<<endl;

}

cout<<" tu : "<<M.tu<<endl;

f1.close();

}

void CopySMatrix(TSMatrix &M);

void AddSMatrix(TSMatrix &M);

void MulSMatrix(TSMatrix &M);

void TransposeSMatrix(TSMatrix M,TSMatrix &T)

{

InitSMatrix(T);

int i,j;

int num[100],cpot[100];

T.mu=M.mu;

T.nu=M.nu;

T.tu=M.tu;

if(T.tu){

for(i=0;i<=M.nu;i++)

{

num[i]=0;

cpot[i]=0;

// cout<<num[i]<<endl;

}

for(i=0;i<M.tu;i++)

{

++num[M.data[i].col];

}

cpot[0]=0;

for(i=1;i<=M.nu;i++)

{

cpot[i]=cpot[i-1]+num[i-1];

}

/\* for(i=0;i<M.nu;i++)

{

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

}

\*/ for(i=0;i<M.tu;i++)

{

j=M.data[i].col;

T.data[cpot[j]].row=M.data[i].col;

T.data[cpot[j]].col=M.data[i].row;

T.data[cpot[j]].e=M.data[i].e;

cpot[j]++;

}

cout<<endl;

for(i=0;i<T.tu;i++)

{

cout<<T.data[i].row<<" "<<T.data[i].col<<" "<<T.data[i].e<<endl;

}

}

}

int main()

{

TSMatrix M,T;

int \*num,\*cpot;

InitSMatrix(M);

ReadSMatrix(M);

TransposeSMatrix(M,T);

return 0;

}

2.

题目：统计一篇英文文章中各单词出现的频度。

输入： 一篇txt格式英文内容

输出： 单词：频度 例如： the: 10; long: 5

算法思想：

1. 将单词存于链表 构造二叉链表 forest[52] 52个字母开头 便于查找

Forest[0]->aa->aaa->and->... 排序存于链表

2.构造串树 a

count

h

然后构成了单词

s

count

count

m

count

l

l

o

i

e

he hello hi his him

在每一个单词的结尾

存单词频度

顺便把地址也存起来了

count

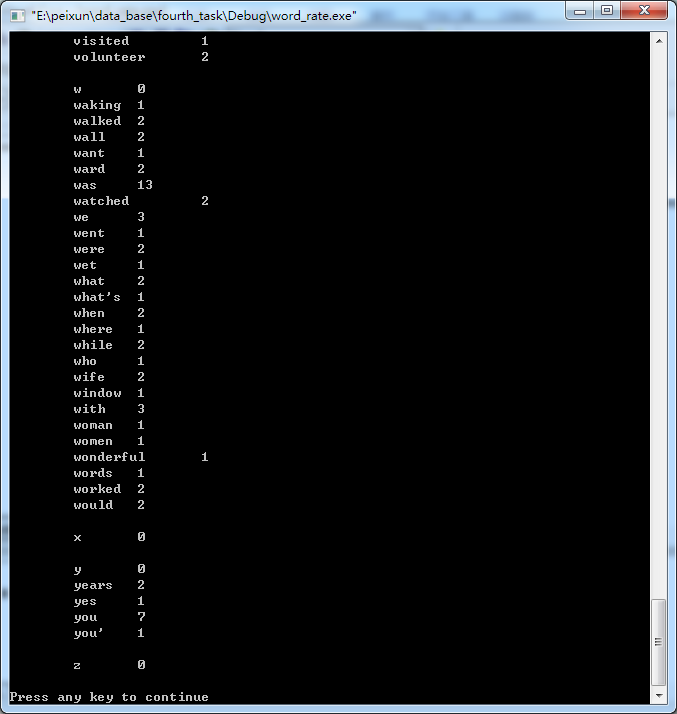
再串树的读取的时候 两种方法

1. 每次讲字母存到string中 每读取

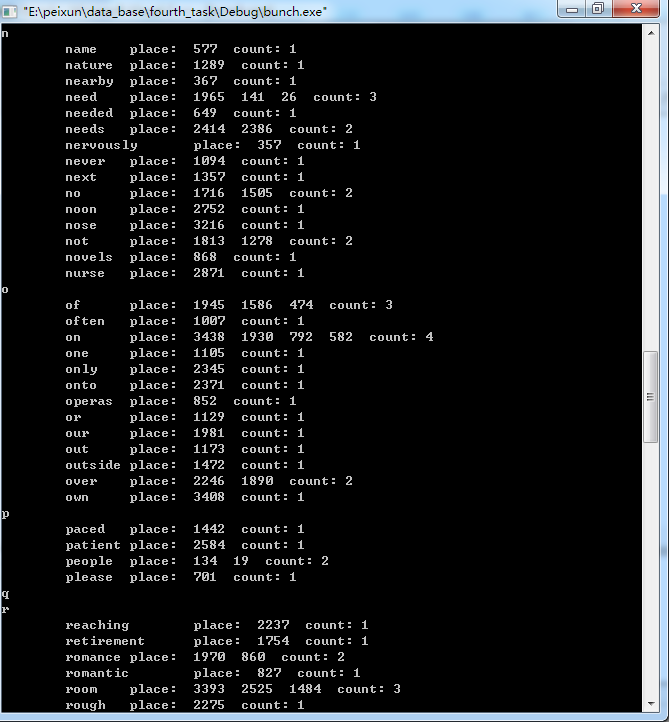
一个节点 string+=alphbet;

2. 通过栈来读取 每次到达叶子结点 通过read\_d\_s(forest) 读取栈元素 从底至上

读取 top元素存储的letter\* 地址 取count 顺便取地址 倒序

1. 运行结果：
2. 链表输出 

会将结果输出到range.txt文件中

1. 串树

附源代码：

1.链表 //完成第一、二、三次上机题

//1．输入稀疏矩阵，建立稀疏矩阵三元组顺序结构，实现转置（1、2）；

//2. 统计一篇英文文章中各单词出现的频度。

// 输入： 一篇txt格式英文内容

// 输出： 单词：频度 例如： the: 10; long: 5

//3. 利用栈操作实现八皇后问题求解 (选做题) 。

#include<iostream>

#include<string>

using namespace std;

#include<fstream>

struct word\_place{

int place;

word\_place \* w\_next;

};

struct word{

int count;

string words;

word \*next,\*pre;

word\_place \*w\_p\_cur\_next,\*w\_p\_head;

};

void man\_txt()

{

fstream f1,f2;

char chr;

f1.open("eng.txt",ios::in);

f2.open("en2.txt",ios::out);

while(!f1.eof())

{

chr=f1.get() ;

if(chr!=','&&chr!='.'&&chr!='\"'&&chr!='?'&&chr!='!')

{

f2<<chr;

// cout<<chr;

}

}

f1.close();

f2.close();

cout<<"end";

}

void display(word \* p)

{

word \*q;

word\_place \*w\_p;

fstream f1;

f1.open("rand.txt",ios::out);

int i=0;

while(i<52)

{

q=&p[i];

while(q)

{

cout<<"\t"<<q->words<<" \t"<<q->count<<" "<<endl;

f1<<"\t"<<q->words<<" \t"<<q->count<<" "<<endl;

w\_p=q->w\_p\_head;

while(w\_p)

{

f1<<"\t\t"<<w\_p->place<<endl;

w\_p=w\_p->w\_next;

}

q=q->next;

}

cout<<endl;

i++;

}

}

int comp()

{

fstream f1;

f1.open("ee2.txt",ios::in);

int i=0;

word w[52];

word \*cur[52];

word\_place \*w\_p;

word \*p,\*q;

int place;

string tmp,tmp\_first;

for(i=0;i<26;i++)

{

w[i].words=(char)(i+'A');

w[i].next=NULL;

w[i].pre=NULL;

w[i].w\_p\_cur\_next=NULL;

w[i].w\_p\_head=NULL;

w[i].count=0;

w[i+26].words=(char)(i+'a');

w[i+26].next=NULL;

w[i+26].pre=NULL;

w[i+26].w\_p\_cur\_next=NULL;

w[i+26].w\_p\_head=NULL;

w[i+26].count=0;

cout<<w[i].words<<" "<<w[i+26].words<<endl;

}

// cout<<(char)('A'+5)<<endl;

// cout<<(a.substr(0,1).compare("a"))<<endl;

while(!f1.eof())

{

place=f1.tellg();

f1>>tmp;

tmp\_first=tmp.substr(0,1);

cout<<tmp<<endl;

if((tmp\_first.compare("A")>=0)&&(tmp\_first.compare("Z")<=0))

{

i=tmp\_first[0]-'A';

}else if((tmp\_first.compare("a")>=0)&&(tmp\_first.compare("z")<=0)){

i=tmp\_first[0]-'a'+26;

}else{

i=-1;

}

if(i!=-1)

{

p = new word;

p->words=tmp;

p->next=NULL;

cout<<"p->words: "<<p->words<<endl;

if(w[i].next==NULL)

{

p->pre=&w[i];

w[i].next=p;

cur[i]=p;

p->count=1;

cout<<"new: "<<p->words<<endl;

//w\_p

w\_p = new word\_place;

w\_p->place=place;

w\_p->w\_next=NULL;

cur[i]->w\_p\_head=w\_p;

cur[i]->w\_p\_cur\_next=w\_p;

// cout<<"ppp"<<endl;

}else{

// cout<<cur[i]->words<<" "<<tmp<<" "<<cur[i]->words.compare(tmp)<<" "<<(cur[i]->words.compare(tmp)<0&&cur[i]->next);

if(cur[i]->words.compare(tmp)>0)

{

/\* cout<<"++++++++++++++++++"<<endl;

cout<<cur[i]->words<<" "<<endl;

cout<<tmp<<endl;

cout<<"++++++++"<<endl;

\*/

while(cur[i]->words.compare(tmp)>0&&cur[i]!=&w[i])

{

cur[i]=cur[i]->pre;

// cout<<"---------"<<endl;

}

/\* cout<<"++++++++++++++++++"<<endl;

cout<<cur[i]->words<<" "<<endl;

cout<<tmp<<endl;

cout<<"++++++++"<<endl;

\*/

}else if(cur[i]->words.compare(tmp)<0){

/\* cout<<"................."<<endl;

cout<<cur[i]->words<<" "<<endl;

cout<<tmp<<endl;

cout<<"..........."<<endl;

\*/

while(cur[i]->words.compare(tmp)<0&&cur[i]->next)

{

cur[i]=cur[i]->next;

}

if(cur[i]->words.compare(tmp)>0)

{

cur[i]=cur[i]->pre;

}

/\* cout<<"..........."<<endl;

cout<<cur[i]->words<<" "<<endl;

cout<<tmp<<endl;

cout<<".........."<<endl;

\*/

}

if(cur[i]->words.compare(tmp)==0)

{

cout<<endl<<endl<<"="<<cur[i]->words<<"===="<<p->words<<endl;

cout<<"pop"<<cur[i]->w\_p\_cur\_next->place<<endl;

w\_p = new word\_place;

w\_p->place=place;

w\_p->w\_next=NULL;

if(cur[i]->w\_p\_cur\_next)

{

cout<<"pop"<<cur[i]->w\_p\_cur\_next->place<<endl;

}

cur[i]->w\_p\_cur\_next->w\_next=w\_p;

cur[i]->w\_p\_cur\_next=w\_p;

cur[i]->count++;

}else{

p->pre=cur[i];

p->next=cur[i]->next;

if(cur[i]->next)

{

cur[i]->next->pre=p;

}

cur[i]->next=p;//插入word节点

//插入w\_p节点

w\_p = new word\_place;

w\_p->place=place;

w\_p->w\_next=NULL;

p->w\_p\_head=w\_p;

p->w\_p\_cur\_next=w\_p;

p->count=1;

cur[i]=p;

cout<<cur[i]->words<<" place: "<<cur[i]->w\_p\_cur\_next->place<<endl;

}

}

}

cout<<"old"<<endl;

}

display(w);

f1.close();

return 0;

}

int main()

{

comp();

return 0;

}

1. 串树：

#include<iostream>

#include<fstream>

#include<string>

using namespace std;

typedef struct stack \* link;

#define maxn 100

struct addr

{

int place;

addr \*next;

};

struct letter

{

char alphabet;

int count;

addr \*addr\_count;

letter \*sibling;

letter \*next;

};

struct play\_stack

{

letter \*p;

char al;

};

play\_stack p\_s[maxn];

struct stack

{

string cur\_str;

letter \*p;

link next;

};

struct show

{

int length;

letter \*p;

};

void init\_letter(letter \*p)

{

p->addr\_count=NULL;

p->alphabet='A';

p->count=0;

p->sibling=0;

p->next=NULL;

}

void init(letter \*p)

{

int i=0;

for(i=0;i<26;i++)

{

p[i].alphabet='A'+i;

p[i].sibling=NULL;

p[i].addr\_count=NULL;

p[i].next=NULL;

p[i].count=0;

p[i+26].alphabet='a'+i;

p[i+26].sibling=NULL;

p[i+26].next=NULL;

p[i+26].count=0;

p[i+26].addr\_count=NULL;

}

}

letter\* set\_p(char chr,letter \*p,letter \*forest)

{

if(chr>='A'&&chr<='Z')

{

p=&forest[((int)chr-'A')];

}else if(chr>='a'&&chr<='z')

{

p=&forest[((int)chr-'a'+26)];

}

return p;

}

letter\* insert(char chr,letter \*p)

{

letter \*cur,\*origin,\*q\_l;

q\_l = new letter;

q\_l->addr\_count=NULL;

q\_l->alphabet=chr;

q\_l->sibling=NULL;

q\_l->next=NULL;

q\_l->count=0;

if(p->next==NULL)

{

p->next=q\_l;

p=q\_l;

}else{

origin=p;

p=p->next;

cur=p;

if(p->alphabet>chr)

{

q\_l->sibling=p;

origin->next=q\_l;

p=q\_l;

}else{

while(p!=NULL)

{

if(p->alphabet<chr)

{

cur=p;

p=p->sibling;

}else{

break;

}

}

if(p==NULL)

{

cur->sibling=q\_l;

p=q\_l;

}else if(p->alphabet>chr)

{

q\_l->sibling=p;

cur->sibling=q\_l;

p=q\_l;

}

}

}

return p;

}

void set\_record(char chr,letter \*p,int place)

{

addr \*q;

q = new addr;

q->next=NULL;

q->place=place;

p->count++;

if(p->addr\_count)

q->next=p->addr\_count->next;

p->addr\_count=q;

}

link push(link Head,string a,letter \*l)

{

link p;

p= new stack;

if(p==NULL)

{

cout<<"\nMemory Error\n";

return Head;

}

p->p=l;

p->next=Head;

p->cur\_str=a;

return p;

}

link pop(link Head)

{

link p;

p=Head;

if(p==NULL)

{

cout<<"\nStack is Empty\n";

}

else{

p=p->next;

}

return p;

}

void travel(letter \*forest)

{

int i=0;

string a;

link Head;

letter \*q\_l;

cout<<"oooo"<<endl;

Head=NULL;

for(i=0;i<52;i++)

{

cout<<forest[i].alphabet<<endl;

// a+=forest[i].alphabet;

if(forest[i].next!=NULL)

{

q\_l=&forest[i];

while(q\_l!=NULL)

{

if(q\_l->sibling!=NULL)

{

letter \*qp;

qp=q\_l->sibling;

Head=push(Head,a,q\_l->sibling);

while(qp)

{

qp=qp->sibling;

}

}

a+=q\_l->alphabet;

if(q\_l->count>0)

{

addr \*p\_arr;

p\_arr=q\_l->addr\_count;

cout<<a<<"\t place: \t";

while(p\_arr)

{

// cout<<"true "<<p\_arr->place;

p\_arr=p\_arr->next;

}

cout<<"\tcount: \t"<<q\_l->count<<endl;

}

if(q\_l->next)

{

q\_l=q\_l->next;

}else{

break;

}

}

while(Head)

{

if(Head->p)

{

}

q\_l=Head->p;

a=Head->cur\_str;

Head=Head->next;

while(q\_l!=NULL)

{

if(q\_l->sibling!=NULL)

{

Head=push(Head,a,q\_l->sibling);

// cout<<"Head"<<Head->p->alphabet<<endl;

}

a+=q\_l->alphabet;

if(q\_l->count>0)

{

cout<<a<<" ";

addr \*p\_arr;

p\_arr=q\_l->addr\_count;

while(p\_arr)

{

cout<<" "<<p\_arr->place<<" ";

p\_arr=p\_arr->next;

}

cout<<q\_l->count<<"\t";

}

q\_l=q\_l->next;

}

}

}

cout<<" "<<endl;

a='.';

}

}

void push\_d\_s(letter \*p,int & top)

{

top++;

p\_s[top].al=p->alphabet;

p\_s[top].p=p;

}

void read\_ps(int top)

{

cout<<"\t";

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

{

cout<<p\_s[i].p->alphabet;

}

addr \*p;

p=p\_s[top].p->addr\_count;

if(p)

cout<<"\tplace: ";

while(p)

{

cout<<" "<<p->place<<" ";

p=p->next;

}

cout<<" count: "<<p\_s[top].p->count<<endl;

}

letter\* pop\_d\_s(int &top)

{

int tmp;

tmp=top;

top--;

return p\_s[tmp].p;

}

void display\_stack(letter \*forest)

{

int top=0;

int cur=0;

letter \*q\_l;

for(int i=0;i<52;i++)

{

top=0;

cout<<forest[i].alphabet<<" "<<endl;

push\_d\_s(&forest[i],top);

q\_l=&forest[i];

if(q\_l->count>0)

{

read\_ps(top);

}

if(forest[i].next)

{

q\_l=forest[i].next;

while(top!=0)

{

while(q\_l!=NULL)

{

push\_d\_s(q\_l,top);

if(q\_l->count>0)

{

read\_ps(top);

}

q\_l=q\_l->next;

}

q\_l=pop\_d\_s(top);

while(q\_l->sibling==NULL&&top!=0)

{

q\_l=pop\_d\_s(top);

}

if(q\_l->sibling!=NULL)

{

q\_l=q\_l->sibling;

}

}

}

}

}

void display(letter \*forest)

{

int i=0;

string a;

letter \*q\_l;

// cout<<"oooo"<<endl;

for(i=0;i<52;i++)

{

cout<<forest[i].alphabet<<" ";

a+=forest[i].alphabet;

if(forest[i].next!=NULL)

{

q\_l=forest[i].next;

while(q\_l->next!=NULL)

{

a+=q\_l->alphabet;

if(q\_l->count>0)

{

cout<<a<<" "<<q\_l->count<<"\t";

}

q\_l=q\_l->next;

}

a+=q\_l->alphabet;

cout<<a<<"\t";

cout<<q\_l->count;

}

cout<<" "<<endl;

a='.';

}

}

void man\_txt()

{

fstream f1;

letter \*p;

addr \*q;

letter \*q\_l;

letter forest[52];

letter \*origin,\*cur;

int place=0;

char chr;

init(forest);

f1.open("1.txt",ios::in);

place=f1.tellg();

chr=f1.get();

while(!f1.eof())

{

while(!((chr>='A'&&chr<='Z')||(chr>='a'&&chr<='z'))&&!f1.eof())

{

place=f1.tellg();

chr=f1.get();

}

if(!f1.eof()&&((chr>='A'&&chr<='Z')||(chr>='a'&&chr<='z')))

p=set\_p(chr,p,forest);

while(!f1.eof()&&((chr>='A'&&chr<='Z')||(chr>='a'&&chr<='z')))

{

chr=f1.get();

if(f1.eof()) break;

if((chr>='A'&&chr<='Z')||(chr>='a'&&chr<='z'))

{

p=insert(chr,p);

}

}

q = new addr;

q->next=NULL;

q->place=place;

p->count++;

addr \*m;

m=p->addr\_count;

if(p->addr\_count!=NULL)

m=m->next;

if(p->addr\_count)

{

q->next=p->addr\_count;

p->addr\_count=q;

// cout<<"1: "<<1<<endl;

}

else

{

p->addr\_count=q;

// cout<<"0: "<<0<<endl;

}

// cout<<place<<endl;

place=f1.tellg();

chr=f1.get();

}

f1.close();

travel(forest);//string输出

display\_stack(forest);//栈输出

cout<<"end";

}

int main()

{

man\_txt();

return 0;

}

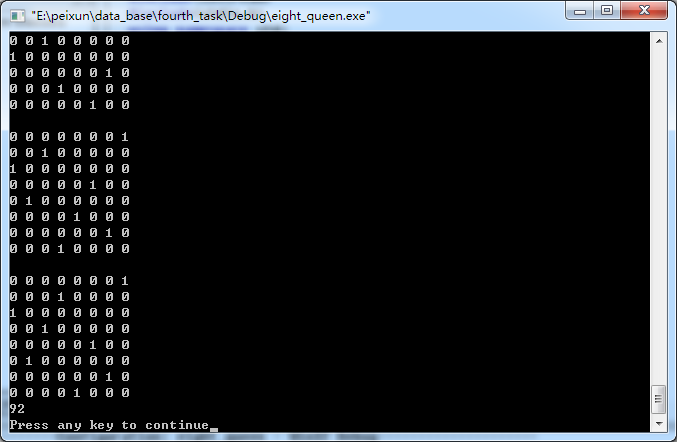
3.

题目：八皇后

算法思想：

栈的基本操作

运行结果：



附源代码：

#include<iostream>

#include<stdio.h>

using namespace std;

#define N 8

struct stack{

int row,col;

};

void pop(stack \*p,int &top,int &row,int &col)

{

if(top>=0&&top<=N)

{

row=p[top].row;

col=p[top].col;

p[top].col=-1;

top--;

}else{

cout<<"wrong top pop"<<endl;

exit(1);

}

}

void push(stack \*p,int &top,int row,int col)

{

if(top<=N&&top>=0)

{

top++;

p[top].row=row;

p[top].col=col;

}else{

cout<<"wrong top"<<endl;

exit(1);

}

}

int back\_search(stack \*p,int top,int row,int col)

{

int i;

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

{

if(p[i].col==col||(row-p[i].row)==(col-p[i].col)||(p[i].row-row)==(col-p[i].col))

{

return 0;

}

}

return 1;

}

void display(stack \*p)

{

int i=0,j;

cout<<endl;

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

{

for(j=0;j<N;j++)

{

if(j!=p[i].col)

{

cout<<0<<" ";

}else{

cout<<1<<" ";

}

}

cout<<endl;

}

}

void queen()

{

stack s[N+1];

int i=0,j=0,top=0;

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

{

s[i].row=0;

s[i].col=-1;

}

i=0;

j=0;

push(s,top,i,j);

i++;

int count=0;

while(!(i==0&&j==N))

{

if(j<N){

if(back\_search(s,top,i,j))

{

push(s,top,i,j);

i++;

j=0;

if(i==N)

{ count++;

display(s);

}

}else{

j++;

}

}else{

if(j>=N){

pop(s,top,i,j);

j++;

}

}

}

cout<<count<<endl;

}

int main()

{

queen();

return 0;

}