算法与数据结构

博客选编

目　录

前言

> 原文出处：算法与数据结构
> 作者：qq_21792169

算法与数据结构

> 算法句数据结构是程序的灵魂，你丢弃了算法也就等于抛弃了优秀程序员的称号

顺序表

著名的计算机科学家N.Wirth教授曾提出一个公式：算法+数据结构=程序

“数组”类型表示顺序存储结构，用指针来表示链式存储结构。指针p指向下一个对象单元，p的值不是一增加1，而是增加对象类型所占的字节数。

一个结构提示类型student,没有定义变量，就不会分配存储单元，不能再程序中直接访问结构体类型名。

线性表是N个具有相同特性的数据元素的有限序列。线性表分为 顺序存储结构和链式存储结构。

顺序表：

```
/*顺序表的建立与输出*//
#include<stdio.h>
#include<malloc.h>
#include<windows.h>
#define maxsize 50
typedef int elemtype;
typedef struct //定义顺序表的存储类型
{
    elemtype data[maxsize];
    int length;
}sqlist;

void createlist(sqlist *&L,elemtype a[],int n)//建立顺序表
{
    int i;
    L=(sqlist *)malloc(sizeof(sqlist));
    for(i=0;i<n;i++)
    L->data[i]=a[i];
    L->length=n;
}
void displist(sqlist *L) //输出顺序表
{
    int i;
    for(i=0;i<L->length;i++)
    printf("%d ",L->data[i]);
    printf("\n");
}

void listempty(sqlist *L) //判断线性表是否为空
{
    int m;
    m=L->length;
```

```
    if(m!=0)
    printf("此线性表不为空\n");
    else
    printf("此为空线性表\n");
}

void listlength(sqlist *L) //求线性表的长度
{
    int m;
    m=L->length;
    printf("此线性表的长度为: %d\n",m);
}

void getelem(sqlist *L) //从顺序表中取值
{
    int i,e;
    printf("请输入需取第几位元素: ");
    scanf("%d",&i);
    if(i<1||i>L->length)
    printf("输入错误");
else
    { e=L->data[i-1];
        printf("取值成功第%d位元素为:%d\n",i,e);
    }
}

void locateelem(sqlist *L) //在顺序表中查找元素
{
    int e,i=0;
    printf("请输入需查找元素:");
    scanf("%d",&e);
    while(i<L->length&&L->data[i]!=e)
    i++;
    if(i>=L->length)
    printf("不存在此元素\n");
    else
    printf("此元素在第%d位\n",i+1);
}

void listinsert(sqlist *&L) //插入元素
{
    int i,j,e;
    printf("请输入插入位置:");
    scanf("%d",&i);
    if(i<1||i>L->length+1)
    printf("输入错误\n");
    else
    {
    printf("请输入需插入元素:");
    scanf("%d",&e);
    i--;
    for(j=L->length;j>i;j--)
    L->data[j]=L->data[j-1];
    L->data[i]=e;
    L->length++;
    printf("插入成功\n");
```

```
    }
}

void listdelete(sqlist *&L) //删除元素
{
    int i,j,e;
    printf("请输入需删除元素位置:");
    scanf("%d",&i);
    if(i<1||i>L->length+1)
    printf("输入错误\n");
    else
    {
    i--;
        e=L->data[i];
        for(j=i+1;j<L->length;j++)
        L->data[j-1]=L->data[j];
        L->length--;
        printf("已删除%d元素\n",e);
    }
}


void main()
{
printf(" ***************欢迎使用顺序表基本运算系统*******************\n");
printf(" *************如有问题请与本人联系**********************\n");
sqlist *q;
int i,m,n=10,a[10];
printf("请输入10个数组元素:\n");
for(i=0;i<10;i++)
scanf("%d",&a[i]);
createlist(q,a,n);
printf("顺序表建立完毕\n");
while(1)
    {
        printf("请选择:");
        printf(" 1.输出链表\n");
        printf(" 2.判断线性表是否为空\n");
        printf(" 3.求线性表的长度\n");
        printf(" 4.从顺序表中取值\n");
        printf(" 5.在顺序表中查找元素\n");
        printf(" 6.插入元素\n");
        printf(" 7.删除元素\n");
        printf(" 8.退出\n");
        scanf("%d",&m);
        switch(m)
            {
            case 1:displist(q);break;
            case 2:listempty(q);break;
            case 3:listlength(q);break;
            case 4:getelem(q);break;
            case 5:locateelem(q);break;
            case 6:listinsert(q);break;
            case 7:listdelete(q);break;
            case 8:exit(0);
            default:printf("输入错误\n");
```

```
            }
        }
    }
```

单链表

单链表：

```
#include<stdio.h>
#include<malloc.h>
#include<windows.h>
typedef int elemtype;
typedef struct LNode //定义单链表存储类型
{
elemtype data;
struct LNode *next;
}linklist;
void creatlistf(linklist *&L ) //建立链表(头插法)
{
linklist *s;
int i;
elemtype a[10];
printf("请输入10个数:\n");
for(i=0;i<10;i++)
scanf("%d",&a[i]);
L=(linklist *)malloc(sizeof(linklist));
L->next=NULL;
for(i=0;i<10;i++)
{
s=(linklist *)malloc(sizeof(linklist));
s->data=a[i];
s->next=L->next;
L->next=s;
}
}
void creatlistr(linklist *&L) //建立链表(尾插法)
{
linklist *s,*r;
int i;
elemtype a[10];
printf("请输入10个数:\n");
for(i=0;i<10;i++)
scanf("%d",&a[i]);
L=(linklist *)malloc(sizeof(linklist));
r=L;
for(i=0;i<10;i++)
{
s=(linklist *)malloc(sizeof(linklist));
s->data=a[i];
r->next=s;
r=s;
}
r->next=NULL;
}
```

```
void displist(linklist *L) //输出单链表
{
linklist *s;
s=L->next;
while(s!=NULL)
{
printf(" %d",s->data);
s=s->next;
}
printf("\n");
}
void listempty(linklist *L) //判断是否为空
{
if(L->next!=NULL)
printf("链表不为空\n");
else
printf("链表为空\n");
}
void listlength(linklist *L) //求链表的长度
{
int n=0;
linklist *p=L;
while(p->next!=NULL)
{
n++;
p=p->next;
}
printf("长度为%d\n",n);
}
void getelem(linklist *L) //取值
{
int m,i=0;
linklist *p=L;
printf("请输入取出元素位置 ");
scanf("%d",&m);
while(i<m&&p!=NULL)
{
i++;
p=p->next;
}
if(p==NULL)
printf("error\n");
else
printf("取值成功 第%d位的元素为 %d\n",m,p->data);
}
void locateelem(linklist *L) //按值查找
{
int m,i=0;
linklist *p=L;
printf("请输入需查找元素值 ");
scanf("%d",&m);
while(p!=NULL&&p->data!=m)
{
i++;
p=p->next;
}
```

```
if(p==NULL)
printf("error\n");
else
printf("元素%d在第%d位\n",m,i);
}
void listinsert(linklist *L) //插入元素
{
int i=0,j,m;
linklist *s,*p;
printf("请输入插入位置:");
scanf("%d",&j);
printf("请输入需插入元素:");
scanf("%d",&m);
s=L;
while(i<j-1 && s!=NULL)
{

s=s->next;
i++;
}
if(s==NULL)
printf("输入错误!\n");
else
{
p=(linklist *)malloc(sizeof(linklist));
p->data=m;
p->next=s->next;
s->next=p;
}
}

void listdelete(linklist *&L) //删除元素
{
int i,j=0,e;
printf("请输入需删除第几个元素:");
scanf("%d",&i);
linklist *s,*q;
s=L;
while(j<i-1&&s!=NULL)
{
j++;
s=s->next;
}
if(s==NULL)
printf("输入错误!\n");
else
{
q=s->next;
if(q!=NULL)
{
e=q->data;
s->next=q->next;
free(q);
printf("成功删除元素%d\n",e);
}
else
```

```
printf("输入错误!\n");
}
}
void destroylist(linklist *&L) //销毁链表
{
char t;
getchar();
printf("确定要销毁链表请输入y否则不销毁: ");
scanf("%c",&t);
if(t=='y')
{
linklist *p=L,*q;
q=p->next;
while(q!=NULL)
{
free(p);
p=q;
q=p->next;
}
free(p);
printf("链表已销毁\n");
exit(0);
}
}



void main()
{ printf(" ***************欢迎使用单链表基本运算系统*******************\n");

printf(" ****************如有问题请与本人联系**************************\n");

linklist *p;
int m,n;
printf("建表方法：1头插法建表, 2尾插法建表\n");
printf("请输入建表方法:");
scanf("%d",&n);
if(n==1)
creatlistf(p); //调用头插法建表函数
else if(n==2)
creatlistr(p); //调用尾插法建表函数
else
{
printf("error\n");
exit(0);
}
printf("链表已建立完毕\n");
while(1)
{
printf("请选择:");
printf(" 1.输出链表\n");
printf(" 2.判断线性表(单链表)是否为空\n");
printf(" 3.求线性表(单链表)的长度\n");
printf(" 4.从线性表(单链表)中取值\n");
printf(" 5.在线性表(单链表)中查找元素\n");
printf(" 6.插入元素\n");
```

```
printf(" 7.删除元素\n");
printf(" 8.销毁链表\n");
printf(" 9.退出\n");
scanf("%d",&m);
switch(m)
{ case 1:displist(p);break;
case 2:listempty(p);break;
case 3:listlength(p);break;
case 4:getelem(p);break;
case 5:locateelem(p);break;
case 6:listinsert(p);break;
case 7:listdelete(p);break;
case 8:destroylist(p);break;
case 9:exit(0);
default:printf("输入错误,请重新输入\n");
}

}
}
```

循环链表

循环链表是指在单链表的最后一个节点链域值不是NULL,而是指向头节点，整个链表形成一个环。h->next=h;

循环链表的操作和单链表基本一致，但是需要在算法中的循环条件p或者p->next是否为空改成是否等于头指针。

下面以循环链表中查找值为x的结点为例来讨论如何实现算法。

```
Lnode *get (Lnode *h ,elemtype x)
{
    Lnode *p;
    p=p->next;
    while(p!=h&&x!=p->data)
    p=p->next;
    return p;
}
```

双向链表

```
#include<stdio.h>
#include<malloc.h>
#include<windows.h>
typedef int elemtype;
typedef struct dnode
{
elemtype data;
struct dnode *prior;
struct dnode *next;
}dlinklist;
void createlistf(dlinklist *&L,elemtype a[],int n) //建立双链表(头插法)
{
dlinklist *s;
int i;
L=(dlinklist *)malloc(sizeof(dlinklist));
L->prior=L->next=NULL;
for(i=0;i<n;i++)
{
s=(dlinklist *)malloc(sizeof(dlinklist));
s->data=a[i];
s->next=L->next;
if(L->next!=NULL)
L->next->prior=s;
L->next=s;
s->prior=L;
}
}
void creatlistr(dlinklist *&L,elemtype a[],int n) //尾插法建表
{
dlinklist *s,*p;
int i;
L=(dlinklist *)malloc(sizeof(dlinklist));
L->prior=L->next=NULL;
p=L;
for(i=0;i<n;i++)
{
s=(dlinklist *)malloc(sizeof(dlinklist));
s->data=a[i];
s->next=p->next;
p->next=s;
s->prior=p;
p=s;
s=p->next;
}
}
void listempty(dlinklist *L) //判断是否为空
{
if(L->next!=NULL)
printf("链表不为空\n");
else
```

```
printf("链表为空\n");
}
int listlength(dlinklist *L) //求链表长度
{
dlinklist *p=L;
int i=0;
while(p!=NULL)
{
i++;
p=p->next;
}
return i-1;
}
void listinsert(dlinklist *&L) //插入元素
{
dlinklist *p=L,*s;
int i,m,a;
printf("请输入需插入元素");
scanf("%d",&a);
printf("请输入需插入元素位置");
scanf("%d",&m);
if(m<0||m>listlength(L))
printf("输入错误\n");
else
{
for(i=0;i<m-1;i++)
p=p->next;
s=(dlinklist *)malloc(sizeof(dlinklist));
s->data=a;
s->next=p->next;
s->next->prior=s;
p->next=s;
s->prior=p;
printf("插入成功\n");
}
}
void listdelete(dlinklist *&L) //删除元素
{
dlinklist *p=L,*q;
int i,m,t;
printf("请输入需删除元素位置 ");
scanf("%d",&m);
if(m<0||m>listlength(L))
printf("输入错误\n");
else
{
for(i=0;i<m-1;i++)
p=p->next;
q=p->next;
p->next=q->next;
q->next->prior=p;
t=q->data;
free(q);
printf("成功删除元素%d\n",t);
}
}
```

```
void getelem(dlinklist *L) //取值
{
dlinklist *p=L;
int i=0,m;
printf("请输入取值元素 ");
scanf("%d",&m);
while(p!=NULL)
{
if(p->data==m)
printf("取值成功 %d元素在第%d位\n",m,i);
p=p->next;
i++;
}
if(p=NULL)
printf("没有此元素\n");
}

void locateelem(dlinklist *L) //查找
{
dlinklist *p=L;
int i=0,m;
printf("请输入需查找位置 ");
scanf("%d",&m);
if(m<=0||m>listlength(L))
printf("输入错误\n");
else
{
for(;i<m;i++)
p=p->next;
printf("第%d位的元素为%d\n",m,p->data);
}
}
void destroylist(dlinklist *&L) //销毁链表
{
dlinklist *p=L,*q;
char m;
getchar();
printf("确认要销毁链表请输入y否则不销毁 请输入:");
scanf("%c",&m);
if(m=='y')
{
q=p->next;
while(q!=NULL)
{
free(p);
p=q;
q=p->next;
}
free(p);
printf("链表已销毁\n");
exit(0);
}
printf("链表未销毁\n");
}
void displist(dlinklist *L) //输出双链表
{
```

```
dlinklist *s;
s=L->next;
while(s!=NULL)
{
printf(" %d",s->data);
s=s->next;
}
printf("\n");
}



void main()
{
printf(" ****************欢迎使用双链表基本运算系统****************\n");
printf(" ****************如有问题欢迎和我联系*****************\n");
dlinklist *p;
int a[5],i,m,t;
printf("请选择建表方式 1用头插法建表, 2用尾插法建表 ");
scanf("%d",&t);
printf("请输入5个数\n");
for(i=0;i<5;i++)
scanf("%d",&a[i]);
if(t==1)
createlistf(p,a,5);
else if(t==2)
creatlistr(p,a,5);
else
printf("输入错误!");
printf("双链表建立完毕\n");
while(1)
{
printf("请选择:");
printf(" 1.输出链表\n");
printf(" 2.判断线性表(双链表)是否为空\n");
printf(" 3.求线性表(双链表)的长度\n");
printf(" 4.从线性表(双链表)中取值\n");
printf(" 5.在线性表(双链表)中查找元素\n");
printf(" 6.插入元素\n");
printf(" 7.删除元素\n");
printf(" 8.销毁链表\n");
printf(" 9.退出\n");
scanf("%d",&m);
switch(m)
{ case 1:displist(p);break;
case 2:listempty(p);break;
case 3: printf("链表长为:%d\n",listlength(p));break;
case 4:getelem(p);break;
case 5:locateelem(p);break;
case 6:listinsert(p);break;
case 7:listdelete(p);break;
case 8:destroylist(p);break;
case 9:exit(0);
default:printf("输入错误,请重新输入\n");
}
}
```

```
}
```

栈的顺序存储

```
#include<stdio.h>
#include<windows.h>
#include<malloc.h>
#define maxsize 50
typedef int elemtype;
typedef struct //定义
{
elemtype data[maxsize];
int top;
}sqstack;
void initstack(sqstack *&s) //初始化
{
s=(sqstack *)malloc(sizeof(sqstack));
s->top=-1;
}
void push(sqstack *&s) //进栈
{
char e;
getchar();
printf("请输入需进栈的元素:");
scanf("%c",&e);
if(s->top>maxsize-1)
printf("栈满，错误!\n");
else
{
s->top++;
s->data[s->top]=e;
printf("进栈成功\n");
}
}
void gettop(sqstack *s) //取栈顶元素
{
char t;
if(s->top==-1)
printf("栈空，取值失败!\n");
else
{
t=s->data[s->top];
printf("取值成功，栈顶元素为:%c\n",t);
}
}
void stackempty(sqstack *s) //判断栈是否为空
{
if(s->top==-1)
printf("栈为空\n");
else
printf("栈不为空\n");
}
void pop(sqstack *&s) //出栈
{
```

```
char e;
if(s->top==-1)
printf("栈为空，出栈失败\n");
else
{
e=s->data[s->top];
s->top--;
printf("出栈成功，出栈元素为:%c\n",e);
}
}
int length(sqstack *s) //求栈长
{
if(s->top==-1)
return(-1);
else
return(s->top+1);
}
void destroy(sqstack *&s) //销毁栈
{
char t;
getchar();
printf("确定要销毁栈请输入y 否则不销毁:");
scanf("%c",&t);
if(t=='y')
{
free(s);
printf("销毁成功\n");
exit(0);
}
else
printf("栈未销毁\n");

}
void output(sqstack *s)
{
int m,n=s->top;
m=length(s)+1;
if(s->top==-1)
printf("栈为空\n");
else
{
printf("出栈序列为:");
for(;m>0;m--)
{
printf(" %c",s->data[s->top]);
s->top--;
}
printf("\n");
s->top=n;
}
}
void main()
{
sqstack *s;
printf(" ************欢迎使用顺序栈运算系统*************\n");
int m;
```

```
initstack(s);
while(1)
{
printf("请选择:");
printf(" 1 进栈\n");
printf(" 2 判断栈是否为空\n");
printf(" 3 取栈顶元素\n");
printf(" 4 出栈\n");
printf(" 5 销毁栈\n");
printf(" 6 求栈的长度\n");
printf(" 7 输出出栈序列\n");
printf(" 8 退出\n");
scanf("%d",&m);
switch(m)
{
case 1:push(s);break;
case 2:stackempty(s);break;
case 3:gettop(s);break;
case 4:pop(s);break;
case 5:destroy(s);break;
case 6:printf("栈的长度为%d\n",length(s));break;
case 7:output(s);break;
case 8:exit(0);
default:printf("输入错误,请重新输入\n");
}
}
}
```

栈的链式存储

```
#include<stdio.h>
#include<windows.h>
#include<malloc.h>
typedef int elemtype;
typedef struct linknode //链表的定义
{
elemtype data;
struct linknode *next;
} listack;
void initstack(listack *&s) //初始化
{
s=(listack *)malloc(sizeof(listack));
s->next=NULL;
}
void push(listack *s) //进栈
{
int e;
listack *p;
printf("请输入进栈元素:");
scanf("%d",&e);
p=(listack *)malloc(sizeof(listack));
p->data=e;
p->next=s->next;
s->next=p;
printf("进栈成功\n");
}
void gettop(listack *s) //取栈顶元素
{
int t;
if(s->next==NULL)
printf("栈空, 取值失败!\n");
else
{
t=s->next->data;
printf("取值成功,栈顶元素为:%d\n",t);
}
}
void stackempty(listack *s) //判断链栈是否为空
{
if(s->next==NULL)
printf("栈为空\n");
else
printf("栈不为空\n");
}
void pop(listack *&s) //出栈
{
listack *p;
int e;
if(s->next!=NULL)
{
```

```
p=s->next;
e=p->data;
s->next=p->next;
free(p);
printf("出栈成功，栈顶元素为:%d\n",e);
}
else
printf("栈为空，出栈失败\n");
}
void destroy(listack *&s)
{
listack *p=s,*q=s->next;
char m;
getchar();
printf("确定要销毁栈，请输入y 否则不销毁!\n");
scanf("%c",&m);
if(m=='y')
{
while(q!=NULL)
{
free(p);
p=q;
q=p->next;
}
free(p);
printf("销毁成功!\n");
exit(0);
}
else
printf("链栈未销毁!\n");
}
void main()
{
listack *s;
int m;
printf(" ******************欢迎使用**********************\n");
initstack(s);
while(1)
{
printf("请选择:");
printf(" 1 进栈\n");
printf(" 2 判断栈是否为空\n");
printf(" 3 取栈顶元素\n");
printf(" 4 出栈\n");
printf(" 5 销毁栈\n");
printf(" 6 退出\n");
scanf("%d",&m);
switch(m)
{
case 1:push(s);break;
case 2:stackempty(s);break;
case 3:gettop(s);break;
case 4:pop(s);break;
case 5:destroy(s);break;
case 6:exit(0);
default:printf("输入错误,请重新输入\n");
```

```
    }
    }
    }
```

队列的顺序存储

```
#include<stdio.h>
#include<windows.h>
#include<malloc.h>
#define maxsize 100
typedef char elemtype;
typedef struct //队列的定义
{
elemtype data[maxsize];
int front,rear;
}sqqueue;
void initqueue(sqqueue *&q) //队列的初始化
{
q=(sqqueue *)malloc(sizeof(sqqueue));
q->front=q->rear=-1;
}
void enqueue(sqqueue *q) //进队
{
char e;
getchar();
printf("请输入需进队元素:");
scanf("%c",&e);
if(q->rear==maxsize-1)
printf("队满，进队失败!\n");
else
{
q->rear++;
q->data[q->rear]=e;
printf("进队成功\n");
}
}
void queueempty(sqqueue *q) //判断队列是否为空
{
if(q->rear==q->front)
printf("队列为空!\n");
else
printf("队列不为空!\n");
}
void dequeue(sqqueue *q) //出队
{
char m;
if(q->rear==q->front)
printf("队空，出队失败!\n");
else
{
q->front++;
m=q->data[q->front];
printf("出队元素为:%c\n",m);
}
}
void destroy(sqqueue *q) //销毁队列
```

```
{
char m;
getchar();
printf("确定要销毁队列请输入y 否则不销毁!\n");
scanf("%c",&m);
if(m=='y')
{
free(q);
printf("队列已销毁!\n");
exit(0);
}
else
printf("队列未销毁!\n");
}
int length(sqqueue *q) //求队列的长度
{
int n=q->rear-q->front;
return n;
}
void main()
{
sqqueue *q;
int m;
printf(" *************欢迎使用队列的运算系统***************\n");
initqueue(q);
while(1)
{
printf("请选择:");
printf("1 进队\n");
printf(" 2 出队\n");
printf(" 3 判断队列是否为空\n");
printf(" 4 销毁队列\n");
printf(" 5 求队列的长度\n");
printf(" 6 退出\n");
scanf("%d",&m);
switch(m)
{
case 1:enqueue(q);break;
case 2:dequeue(q);break;
case 3:queueempty(q);break;
case 4:destroy(q);break;
case 5:printf("队列的长度为:%d\n",length(q));break;
case 6:exit(0);
default:printf("输入错误, 请从新输入!\n");
}
}
}
```

队列的链式存储

```
#include<stdio.h>
#include<malloc.h>
#include<windows.h>
typedef int elemtype;
typedef struct qnode //数据节点的定义
{
elemtype data;
struct qnode *next;
}qnode;
typedef struct //链队的定义
{
qnode *front;
qnode *rear;
}liqueue;
void initqueue(liqueue *&q) //初始化
{
q=(liqueue *)malloc(sizeof(liqueue));
q->front=q->rear=NULL;
}
void enqueue(liqueue *q) //进队
{
int e;
qnode *p;
p=(qnode *)malloc(sizeof(qnode));
printf("请输入需进队元素:");
scanf("%d",&e);
p->data=e;
p->next=NULL;
if(q->rear==NULL)
{
q->front=q->rear=p;
printf("进队成功!\n");
}
else
{
q->rear->next=p;
q->rear=p;
printf("进队成功!\n");
}
}
void queueempty(liqueue *q) //判断队列是否为空
{
if(q->rear==NULL)
printf("队列为空!\n");
else
printf("队列不为空!\n");
}
void dequeue(liqueue *&q) //出队列
{
int e;
```

```
qnode *t;
if(q->rear==NULL)
printf("队列为空，出队失败!\n");
else
{
t=q->front;
e=t->data;
if(q->front==q->rear)
q->front=q->rear=NULL;
else
q->front=q->front->next;
free(t);
printf("元素%d出队列成功!\n",e);
}
}
void destroyqueue(liqueue *q) //销毁链队
{
qnode *p=q->front,*r;
char t;
getchar();
printf("确定要销毁链队请输入y，否则不销毁!\n");
scanf("%c",&t);
if(t=='y')
{
if(p!=NULL)
{
r=p->next;
while(r!=NULL)
{
free(p);
p=r;
r=p->next;
}
}
free(p);
free(q);
printf("销毁成功\n");
exit(0);
}
else
printf("链队未销毁!\n");
}
void main()
{
liqueue *q;
printf(" ***************欢迎使用链队的运算系统******************\n");
int m;
initqueue(q);
while(1)
{
printf("请选择:");
printf("1 进队列\n");
printf(" 2 出队列\n");
printf(" 3 判断队列是否为空\n");
printf(" 4 销毁链队\n");
printf(" 5 退出\n");
```

```
scanf("%d",&m);
switch(m)
{
case 1:enqueue(q);break;
case 2:dequeue(q);break;
case 3:queueempty(q);break;
case 4:destroyqueue(q);break;
case 5:exit(0);
default:printf("输入错误请重新输入!\n");
}
}
}
```

顺序串

```
#include<stdio.h>
#include<windows.h>
#define maxsize 100
typedef struct //非紧缩格式的顺序串的定义
{
char data[maxsize];
int length;
}sqstring;
void strassign(sqstring &s) //将字符串复制给串
{
char a[100];
int i;
getchar();
printf("请输入一个字符串:");
gets(a);
for(i=0;a[i]!='\0';i++)
s.data[i]=a[i];
s.length=i;
}
void dispstr(sqstring s) //输出串的所有元素
{
int i;
if(s.length<=0)
printf("串为空，输出失败!\n");
else
{
if(s.length>0)
{
for(i=0;i<s.length;i++)
printf(" %c",s.data[i]);
}
printf("\n");
}
}
void strcopy(sqstring &s,sqstring t) //串与串的复制
{
int i;
for(i=0;i<t.length;i++)
s.data[i]=t.data[i];
s.length=t.length;
}
int strequal(sqstring s) //判断串相等
{
sqstring t;
strassign(t);
int i,m=1;
if(s.length!=t.length)
m=0;
else
for(i=0;i<s.length;i++)
```

```
if(s.data[i]!=t.data[i])
m=0;
if(m==1)
printf("串相等\n");
else
printf("串不相等\n");
return m;
}
void strlength(sqstring s) //求串长
{
printf("s.length=%d\n",s.length);
}
sqstring concate(sqstring s) //串的连接
{
sqstring t,str;
strassign(t);
int i,j;
if(s.length==0)
str.length=0;
else
{
str.length=s.length+t.length;
for(i=0;i<s.length;i++)
str.data[i]=s.data[i];
for(i=0;i<t.length;i++)
str.data[s.length+i]=t.data[i];
}
return str;
}
void substr(sqstring s) //求子串
{
sqstring str;
int i,j,k;
printf("请输入第i个字符开始的连续j个字符组成的子串的i、j值\n");
printf("i=");
scanf("%d",&i);
printf("j=");
scanf("%d",&j);
if(s.length==0)
printf("串为空!\n");
else if(i<=0||i>s.length||j<0||i+j-1>s.length)
printf("输入错误!\n");
else
{
str.length=j;
for(k=i-1;j>0;j--,k++)
str.data[k-i+1]=s.data[k];
printf("所求子串为:");
dispstr(str);
}
}
void insstr(sqstring &s) //将串s2插到串s的第i个字符中
{
sqstring s2,str;
int i,j;
str.length=0;
```

```
strassign(s2);
printf("请输入需把串插入的位置:");
scanf("%d",&i);
if(i<0||i>s.length+1)
printf("输入错误!\n");
else
{
for(j=0;j<i-1;j++)
str.data[j]=s.data[j];
for(j=0;j<s2.length;j++)
str.data[j+i-1]=s2.data[j];
for(j=i-1;j<s.length;j++)
str.data[j+s2.length]=s.data[j];
str.length=s.length+s2.length;
strcopy(s,str);
printf("插入后的串为:");
dispstr(s);
}
}
void delstr(sqstring &s) //从串s中删去第i个字符开始的长度为j的子串
{
int i,j,k;
printf("删除第i个字符开始的长度为j的子串\n");
printf("i=");
scanf("%d",&i);
printf("j=");
scanf("%d",&j);
if(s.length==0)
printf("串为空!\n");
else if(i<=0||i>s.length||i+j-1>s.length)
printf("输入错误!\n");
else
{
k=j;
for(;j>=0;j--,i++)
s.data[i-1]=s.data[i+k-1];
s.length=s.length-k;
printf("删除后的串为: ");
dispstr(s);
}
}
void repstr(sqstring &s) //将第i个字符开始的j个字符够成的子串用串t替换
{
sqstring t,str;
int i,j,k;
printf("s串中将第i个字符开始的j个字符够成的子串用串t替换\n");
printf("i=");
scanf("%d",&i);
printf(" j=");
scanf("%d",&j);
strassign(t);
if(i<0||i>s.length||i+j-1>s.length)
printf("输入错误!\n");
else
{
str.length=0;
```

```
for(k=0;k<i-1;k++)
str.data[k]=s.data[k];
for(k=0;k<t.length;k++)
str.data[k+i-1]=t.data[k];
for(k=i+j-1;k<s.length;k++)
str.data[k-j+t.length]=s.data[k];
str.length=s.length+t.length-j;
strcopy(s,str);
printf("替换后的串为:");
dispstr(s);
}
}
void index(sqstring s,sqstring t) //串的模式匹配(bf算法)
{
int i=0,j=0;
while (i<s.length && j<t.length)
{
if (s.data[i]==t.data[j])
{
i++;
j++;
}
else
{
i=i-j+1;
j=0;
}
}
if (j>=t.length)
printf("匹配成功在第%d位\n\n",i-t.length+1);
else
printf("匹配不成功!\n");
}
void main()
{
sqstring s,t;
int m;
printf(" ******************欢迎使用顺序串的运算系统*************\n");
while(1)
{
printf("请选择:");
printf("1 将字符串复制给串s\n");
printf(" 2 串与串的复制\n");
printf(" 3 判断串相等\n");
printf(" 4 求串的长度\n");
printf(" 5 串的连接\n");
printf(" 6 求子串\n");
printf(" 7 将串s2插到串s的第i个字符位置\n");
printf(" 8 从串s中删去从第i个字符开始的长度为j的子串\n");
printf(" 9 将第i个字符开始的j个字符够成的子串用串t替换\n");
printf(" 10 串的模式匹配\n");
printf(" 11 输出串的所有元素\n");
printf(" 12 退出\n");
scanf("%d",&m);
switch(m)
{
```

```
case 1:strassign(s);
printf("复制成功\n");break;
case 2:strassign(t);
strcopy(s,t);
printf("复制成功\n");break;
case 3:strequal(s);break;
case 4:strlength(s);break;
case 5:dispstr(concate(s));break;
case 6:substr(s);break;
case 7:insstr(s);break;
case 8:delstr(s);break;
case 9:repstr(s);break;
case 10:strassign(t);index(s,t);break;
case 11:printf("s: ");dispstr(s);break;
case 12:printf("谢谢使用!\n");exit(0);break;
default:printf("输入错误!\n");
}
}
}
```

稀疏矩阵

```
#include<stdio.h>
#include<windows.h>
#define m 3 //行数
#define n 2 //列数
#define maxsize 50
typedef int elemtype;
typedef struct
{
int r;
int c;
elemtype d;
}tupnode;
typedef struct
{
int rows;
int cols;
int nums;
tupnode data[maxsize];
}tsmatrix;

void creatmat(tsmatrix &t,elemtype a[m][n]) //稀疏矩阵创建三元组表示
{
int i,j;
t.rows=m;t.cols=n;t.nums=0;
for(i=0;i<m;i++)
{
for(j=0;j<n;j++)
if(a[i][j]!=0)
{
t.data[t.nums].r=i;
t.data[t.nums].c=j;
t.data[t.nums].d=a[i][j];
t.nums++;
}
}
}
int value(tsmatrix &t,elemtype x,int i,int j) //三元组元素赋值
{
int k=0,k1;
if(i>=t.rows||j>=t.cols)
return 0;
while(k<t.nums&&i>t.data[k].r)
k++;
while (k<t.nums&&i==t.data[k].r&&j>t.data[k].c)
k++;
if(t.data[k].r==i&&t.data[k].c==j)
t.data[k].d=x;
else
{
for(k1=t.nums-1;k1>=k;k1--)
```

```
{
t.data[k1+1].r=t.data[k1].r;
t.data[k1+1].c=t.data[k1].c;
t.data[k1+1].d=t.data[k1].d;
}
t.data[k].r=i;
t.data[k].c=j;
t.data[k].d=x;
t.nums++;
}
return 1;
}

int assign(tsmatrix t,elemtype &x,int i,int j) //将指定位置的元素值赋给变量
{
int k=0;
if(i>=t.rows||j>=t.cols)
return 0;
while (k<t.nums&&i>t.data[k].r)
k++;
while (k<t.nums&&i==t.data[k].r&&j>t.data[k].c)
k++;
if(t.data[k].r==i&&t.data[k].c==j)
x=t.data[k].d;
else
x=0;
return 1;
}



void dispmat(tsmatrix t) //输出三元组
{
int i;
if(t.nums<=0)
return;
printf("\t%d\t%d\t%d\n",t.rows,t.cols,t.nums);
printf("\t------------------\n");
for(i=0;i<t.nums;i++)
printf("\t%d\t%d\t%d\n",t.data[i].r,t.data[i].c,t.data[i].d);
}

void trantat(tsmatrix t,tsmatrix &tb) //矩阵转置
{
int p,q=0,v;
tb.rows=t.cols;
tb.cols=t.rows;
tb.nums=t.nums;
if(t.nums!=0)
{
for(v=0;v<t.cols;v++)
for(p=0;p<t.nums;p++)
if(t.data[p].c==v)
{
tb.data[q].r=t.data[p].c;
tb.data[p].c=t.data[p].r;
```

```
tb.data[q].d=t.data[p].d;
q++;
}
}
printf("转置后的三元组为:\n");
dispmat(tb);
}
void main()
{
tsmatrix t,tb;
elemtype a[3][2],x;
int i,j,w,g,h;
printf(" *************欢迎使用稀疏矩阵基本运算系统***************\n");
printf("请输入%d个数\n",m*n);
for(i=0;i<m;i++)
for(j=0;j<n;j++)
scanf("%d",&a[i][j]);
creatmat(t,a);
while(1)
{
printf("请选择:");
printf("1 三元组元素赋值\n");
printf(" 2 将指定位置的元素值赋给变量\n");
printf(" 3 矩阵转置\n");
printf(" 4 输出三元组\n");
printf(" 5 退出\n");
scanf("%d",&w);
switch(w)
{
case 1:printf("请输入所赋值:");
scanf("%d",&x);
printf("\n请输入插入第几行:");
scanf("%d",&g);
printf("\n请输入插入第几列:");
scanf("%d",&h);
if(value(t,x,g,h))
printf("赋值成功\n");
else
printf("赋值失败\n");
break;
case 2:printf("\n请需取元素所在行:");
scanf("%d",&g);
printf("\n请输入需取元素所在列:");
scanf("%d",&h);
if(assign(t,x,g,h))
printf("取值成功, 元素为:%d\n",x);
else
printf("取值失败\n");
break;
case 3:trantat(t,tb);break;
case 4:dispmat(t);break;
case 5:exit(0);
default:printf("输入错误!\n");
}
}
}
```

广义表

```
#include<stdio.h>
#include<malloc.h>
#include<windows.h>
typedef char elemtype;
typedef struct lnode //广义表的定义
{
int tag;
union
{
elemtype data;
struct lnode *sublist;
}val;
struct lnode *link;
}glnode;

glnode *creategl(char *&s) //建立广义表的链式存储结构
{
glnode *g;
char ch=*s++;
if(ch!='\0')
{
g=(glnode *)malloc(sizeof(glnode));
if(ch=='(')
{
g->tag=1;
g->val.sublist=creategl(s);
}
else if(ch==')')
g=NULL;
else if(ch=='#')
g=NULL;
else
{
g->tag=0;
g->val.data=ch;
}
}
else
g=NULL;
ch=*s++;
if(g!=NULL)
if(ch==',')
g->link=creategl(s);
else
g->link=NULL;
return g;
}

int gllength(glnode *g) //求广义表的长度
{
```

```
int n=0;
glnode *g1;
g1=g->val.sublist;
while(g1!=NULL)
{
n++;
g1=g1->link;
}
return n;
}

int gldepth(glnode *g) //求广义表的深度
{
glnode *g1;
int max=0,dep;
if(g->tag==0)
return 0;
g1=g->val.sublist;
if(g1==NULL)
return 1;
while(g1-NULL)
{
if(g1->tag==1)
{
dep=gldepth(g1);
if(dep>max)
max=dep;
}
g1=g1->link;
}
return(max+1);
}

elemtype maxatom(glnode *g) //求广义表中的最大原子
{
elemtype max1,max2;
if(g!=NULL)
{
if(g->tag==0)
{
max1=maxatom(g->link);
return(g->val.data>max1?g->val.data:max1);
}
else
{
max1=maxatom(g->val.sublist);
max2=maxatom(g->link);
return(max1>max2?max1:max2);
}
}
else
return 0;
}
void dispgl(glnode *g) //输出广义表
{
if(g!=NULL)
```

```
{
if(g->tag==0)
printf("%c",g->val.data);
else
{
printf("(");
if(g->val.sublist==NULL)
printf("#");
else
dispgl(g->val.sublist);
printf(")");
}
if(g->link!=NULL)
{
printf(",");
dispgl(g->link);
}
}
}

void main()
{
glnode *g;
char a[100],*p=a;
int m;
printf(" **************欢迎使用广义表的运算系统****************\n");
printf("请输入一个广义表:(例如(b,(b,a,(#),d),((a,b),c,((#)))))\n");
gets(a);
g=creategl(p);
printf("广义表已建好\n");
while(1)
{
printf("请选择:");
printf(" 1求广义表的长度\n");
printf(" 2求广义表的深度\n");
printf(" 3输出广义表\n");
printf(" 4求广义表中的最大原子\n");
printf(" 5退出\n");
scanf("%d",&m);
switch(m)
{
case 1:printf("广义表的长度为:%d\n",gllength(g));break;
case 2:printf("广义表的深度为:%d\n",gldepth(g));break;
case 3:printf("广义表为:");dispgl(g);printf("\n");break;
case 4:printf("广义表中的最大原子为:%c\n",maxatom(g));break;
case 5:exit(0);
default:printf("输入错误\n");
}
}
}
```

二叉树

```
#include<stdio.h>
#include<windows.h>
#include<malloc.h>
#define maxsize 20
typedef int elemtype;
typedef struct node //定义
{
elemtype data;
struct node *lchild;
struct node *rchild;
}btnode;
void createbtnode(btnode *&b,char *str) //创建二叉树
{
btnode *st[maxsize],*p=NULL;
int top=-1,k,j=0;
char ch;
b=NULL;
ch=str[j];
while(ch!='\0')
{
switch(ch)
{
case '(':top++;st[top]=p;k=1;break;
case ')':top--;break;
case ',':k=2;break;
default:p=(btnode *)malloc(sizeof(btnode));
p->data=ch;p->lchild=p->rchild=NULL;
if(b==NULL)
b=p;
else
{
switch(k)
{
case 1:st[top]->lchild=p;break;
case 2:st[top]->rchild=p;break;
default:printf("错误!\n");
}
}
}
j++;
ch=str[j];
}
}
btnode *findnode(btnode *b,elemtype x) //查找节点
{
btnode *p;
if(b==NULL)
return NULL;
else if(b->data==x)
return b;
```

```
else
{
p=findnode(b->lchild,x);
if(p!=NULL)
return p;
else
return findnode(b->rchild,x);
}
}
btnode *lchildnode(btnode *p) //找左孩子
{
return p->lchild;
}
btnode *rchildnode(btnode *p) //找右孩子
{
return p->rchild;
}
int btnodeheight(btnode *b) //求高度
{
int lchildh,rchildh;
if(b==NULL)
return 0;
else
{
lchildh=btnodeheight(b->lchild);
rchildh=btnodeheight(b->rchild);
return(lchildh>rchildh)?(lchildh+1):(rchildh+1);
}
}
void dispbtnode(btnode *b) //输出二叉树
{
if(b!=NULL)
{
printf("%c",b->data);
if(b->lchild!=NULL||b->rchild!=NULL)
{
printf("(");
dispbtnode(b->lchild);
if(b->rchild!=NULL)
printf(",");
dispbtnode(b->rchild);
printf(")");
}
}
}


int Nodes(btnode *b)//求二叉树b的节点个数
{
int num1,num2;
if (b==NULL)
return 0;
else if (b->lchild==NULL && b->rchild==NULL)
return 1;
else
{
```

```
num1=Nodes(b->lchild);
num2=Nodes(b->rchild);
return (num1+num2+1);
}
}
int LeafNodes(btnode *b)//求二叉树b的叶子节点个数
{
int num1,num2;
if (b==NULL)
return 0;
else if (b->lchild==NULL && b->rchild==NULL)
return 1;
else
{
num1=LeafNodes(b->lchild);
num2=LeafNodes(b->rchild);
return (num1+num2);
}
}
void TravLevel(btnode *b) //层次遍历
{
btnode *Qu[maxsize];//定义循环队列
int front,rear;//定义队首和队尾指针
front=rear=0;//置队列为空队列
if (b!=NULL)
printf("%c ",b->data);
rear++;//节点指针进入队列
Qu[rear]=b;
while (rear!=front)//队列不为空
{
front=(front+1)%maxsize;
b=Qu[front];//队头出队列
if (b->lchild!=NULL)//输出左孩子,并入队列
{
printf("%c ",b->lchild->data);
rear=(rear+1)%maxsize;
Qu[rear]=b->lchild;
}
if (b->rchild!=NULL)//输出右孩子,并入队列
{
printf("%c ",b->rchild->data);
rear=(rear+1)%maxsize;
Qu[rear]=b->rchild;
}
}
printf("\n");
}
void PreOrder(btnode *b) //先序遍历的递归算法
{
if (b!=NULL)
{
printf("%c ",b->data);//访问根节点
PreOrder(b->lchild);//递归访问左子树
PreOrder(b->rchild);//递归访问右子树
}
}
```

```
void PreOrder1(btnode *b) //先序遍历的非递归算法
{
btnode *St[maxsize],*p;
int top=-1;
if (b!=NULL)
{
top++;//根节点入栈
St[top]=b;
while (top>-1)//栈不为空时循环
{
p=St[top];//退栈并访问该节点
top--;
printf("%c ",p->data);
if (p->rchild!=NULL)//右孩子入栈
{
top++;
St[top]=p->rchild;
}
if (p->lchild!=NULL)//左孩子入栈
{
top++;
St[top]=p->lchild;
}
}
printf("\n");
}
}
void InOrder(btnode *b) //中序遍历的递归算法
{
if (b!=NULL)
{
InOrder(b->lchild);//递归访问左子树
printf("%c ",b->data);//访问根节点
InOrder(b->rchild);//递归访问右子树
}
}
void InOrder1(btnode *b) //中序遍历的非递归算法
{
btnode *St[maxsize],*p;
int top=-1;
if (b!=NULL)
{
p=b;
while (top>-1 || p!=NULL)
{
while (p!=NULL)
{
top++;
St[top]=p;
p=p->lchild;
}
if (top>-1)
{
p=St[top];
top--;
printf("%c ",p->data);
```

```
p=p->rchild;
}
}
printf("\n");
}
}
void PostOrder(btnode *b) //后序遍历的递归算法
{
if (b!=NULL)
{
PostOrder(b->lchild);//递归访问左子树
PostOrder(b->rchild);//递归访问右子树
printf("%c ",b->data);//访问根节点
}
}
void PostOrder1(btnode *b) //后续遍历的非递归算法
{
btnode *St[maxsize];
btnode *p;
int flag,top=-1;//栈指针置初值
if (b!=NULL)
{
do
{
while (b!=NULL)//将t的所有左节点入栈
{
top++;
St[top]=b;
b=b->lchild;
}
p=NULL;//p指向当前节点的前一个已访问的节点
flag=1;
while (top!=-1 && flag)
{
b=St[top];//取出当前的栈顶元素
if (b->rchild==p)//右子树不存在或已被访问,访问之
{
printf("%c ",b->data);//访问*b节点
top--;
p=b;//p指向则被访问的节点

}



else
{
b=b->rchild;//t指向右子树
flag=0;
}
}
} while (top!=-1);
printf("\n");
}
}
void DestroyBTNode(btnode *&b) //销毁二叉树
```

```
{
if (b!=NULL)
{
DestroyBTNode(b->lchild);
DestroyBTNode(b->rchild);
free(b);
}

}



void main()
{
int hight,n;
char ch,str[100];
btnode *b,*p,*p1,*p2;
printf(" ******************欢迎使用二叉树基本运算系统*******************\n");

printf("请输入一个广义表\n(例如:A(B(D(,I),E(J,K(H))),C(F)))\n");
gets(str);
createbtnode(b,str);
while(1)
{
printf("请选择:\n");
printf(" 1 求树的高度\n");
printf(" 2 求节点的孩子\n");
printf(" 3 输出二叉树\n");
printf(" 4 求二叉树的节点数\n");
printf(" 5 求二叉树的叶子节点数\n");
printf(" 6 层次遍历序列\n");
printf(" 7 先序遍历序列\n");
printf(" 8 中序遍历序列\n");
printf(" 9 后续遍历序列\n");
printf(" 10 释放二叉树并退出\n");
printf(" 11 退出\n");
scanf("%d",&n);
switch(n)
{
case 1:printf("树的高度为:%d\n",btnodeheight(b));break;
case 2:getchar();printf("请输入需查找的节点:");
scanf("%c",&ch);
p=findnode(b,ch);
p1=lchildnode(p);
p2=rchildnode(p);
if(p1!=NULL)
printf("左孩子为:%c\n",p1->data);
else
printf("没有左孩子\n");
if(p2!=NULL)
printf("右孩子为:%c\n",p2->data);
else
printf("没有右孩子\n");
break;
case 3:dispbtnode(b);
printf("\n");
```

```
break;
case 4:printf("二叉树的节点个数为:%d个\n",Nodes(b));break;
case 5:printf("二叉树的叶子节点数为%d个\n",LeafNodes(b));break;
case 6:printf("层次遍历序列:");TravLevel(b);break;
case 7:printf("先序遍历序列:\n");
printf(" 递归算法:");PreOrder(b);printf("\n");
printf(" 非递归算法:");PreOrder1(b);break;
case 8:printf("中序遍历序列:\n");
printf(" 递归算法:");InOrder(b);printf("\n");
printf(" 非递归算法:");InOrder1(b);break;
case 9:printf("后序遍历序列:\n");
printf(" 递归算法:");PostOrder(b);printf("\n");
printf(" 非递归算法:");PostOrder1(b);break;
case 10:DestroyBTNode(b);printf("二叉树已销毁!\n");exit(0);break;
case 11:exit(0);
default:printf("输入错误\n");
}
}
}
```

链表的实战讲解（综合以前的基础）

这是前面我讲得算法与数据结构中链表的综合，如果这里不明白请看前面的基础知识：链接地址。

```
#include <stdio.h>

#include <stdlib.h>
#include <string.h>



/* 定义一个结构体 */

typedef struct NAME{
char *name;
struct NAME *pre;
struct NAME *next;
}T_Name, *PT_Name;

static PT_Name   g_ptNameHead;    /* 定义链表头 */

void add_name(PT_Name ptNew)
{
PT_Name ptCur;

if (g_ptNameHead == NULL)
{
g_ptNameHead = ptNew;
}
else
{
ptCur = g_ptNameHead;
while (ptCur->next)
{
ptCur = ptCur->next;
}
ptCur->next = ptNew;
ptNew->pre  = ptCur;
}
}

void del_name(PT_Name ptDel)
{
PT_Name ptCur;
PT_Name ptPre;
PT_Name ptNext;

if (g_ptNameHead == ptDel)   /* 如果链表头等于当前删除的链表 */
{
```

```
g_ptNameHead = ptDel->next;
/* 释放 */
return;
}
else
{
ptCur = g_ptNameHead->next;
while (ptCur)
{
if (ptCur == ptDel)
{
/* 从链表中删除 */
ptPre  = ptCur->pre;
ptNext = ptCur->next;
ptPre->next = ptNext;
if (ptNext)
{
ptNext->pre = ptPre;
}
break;
}
else
{
ptCur = ptCur->next;
}
}
}

free(ptDel->name);
free(ptDel);
}



void add_one_name()

{
PT_Name ptNew;
char *str;
char name[128];

printf("enter the name:");
scanf("%s", name);

str  = malloc(strlen(name) + 1);

/* name是一个局部变量，用来存放名字，当这个函数结束的时候，该内存就得释放，所以我们得单独分配一块内来存放这个name ,下面还得用malloc来分配一个结构体大的内存空间，记住我们定义结构体的时候不能添加static，因为定义结构提示不会分配内存空间的，他定义的只是这种类型，所以我们增加的时候一定要记得分配内存*/

strcpy(str, name);
ptNew = malloc(sizeof(T_Name));
ptNew->name = str;
ptNew->pre  = NULL;
ptNew->next = NULL;
```

```
add_name(ptNew);
}

PT_Name get_name(char *name)
{
PT_Name ptCur;
if (g_ptNameHead == NULL)
{
return NULL;
}
else
{
ptCur = g_ptNameHead;
do {
if (strcmp(ptCur->name, name) == 0)
return ptCur;
else
ptCur = ptCur->next;
}while (ptCur);
}



return NUL;

}

void del_one_name()
{
PT_Name ptFind;
char name[128];

printf("enter the name:");
scanf("%s", name);

ptFind = get_name(name);

if (ptFind == NULL)
{
printf("do not have this name\n");
return ;
}

del_name(ptFind);
}

void list_all_name(void)
{
PT_Name ptCur;
int i = 0;
ptCur = g_ptNameHead;
while (ptCur)
{
printf("%02d : %s\n", i++, ptCur->name);
ptCur = ptCur->next;
```

```
}
}

int main(int argc, char **argv)
{
char c;
while (1)
{
printf("<l> List all the names\n");
printf("<a> add one name\n");
printf("<d> del one name\n");
printf("<x> exit\n");
printf("Enter the choise: ");

c = getchar();

switch (c)
{
case 'l':
{
list_all_name();
break;
}
case 'a':

{

add_one_name();

break;
}
case 'd':
{
del_one_name();
break;
}
case 'x':
{
return 0;
break;
}
default:
{
break;
}
}
}

return 0;
}
```