**5.**

void mydelete(linklist\* head,linklist\* tail,int i){ //5.删除节点

linklist pcur,ppre;

pcur=ppre=\*head;

if(pcur!=NULL){ //头指针不为空

if(pcur->num==i){ //头指针为待删除节点

\*head=pcur->next; //改变为下一节点

if(\*head==NULL) //删除后节点为空

\*tail=NULL;

free(pcur);

}else{

while(pcur!=NULL){ //遍历中间寻找待删节点

if(pcur->num==i){

ppre->next=pcur->next;

free(pcur);

break;

}else{

ppre=pcur;

pcur=pcur->next;

}

if(ppre->next==NULL) //到末端，改变尾指针

\*tail=ppre;

if(pcur==NULL) //遍历到末尾未找到

printf("no this node\n");

}

}

}else

printf("list is NULL!\n"); //空链表

}

**7.**

void reverse\_d\_link(linkslist\* head,linkslist\*tail){ //逆置节点

linkslist tou,wei;

int temp=0;

tou=(\*head);

wei=(\*tail);

while(tou>=wei){

temp=tou->num;

tou->num=wei->num;

wei->num=temp;

tou=tou->next;

wei=wei->pre;

}

printf("该链表已经逆置：\n");

}

**8.**

void find\_reciprocal\_4(linkslist tail){ //寻找倒数第四个节点

int count=0;

while(count<3){

tail=tail->pre;

count++;

}

if(count<3)

printf("该链表长度小于4！无法查找！\n");

else

printf("倒数第四个数字为：%d\n",tail->num);

}

**9.**

void find\_mid\_link(linkslist head,linkslist tail){ //寻找中间节点

while(head>tail){

head=head->next;

tail=tail->pre;

}

if(head==tail)

printf("该链表长度为奇数，中间节点为1个 ：%d \n",head->num);

else

printf("该链表长度为偶数，中间节点为2个：%d %d \n",head->num,tail->num);

}

**11.**

void twolist\_has\_same\_node(linklist head1,linklist head2){ //11.判断两个链表是否相交

linklist j;

j=head2;

for(;head1!=NULL;head1=head1->next){

for(j=head2;j!=NULL;j=j->next)

if(j==head1){

printf("交点为 ：%d",j->num);

}

}

}

**12.**

void del\_sameelement\_in\_list(linklist \*head,linklist \*tail){ //12.删除相同元素

linklist i,j,jp;

int tag=0;

i=\*head;

j=i->next;

jp=i->next;

if(i==NULL&&i->next==NULL){

printf("链表小于等于1，不能删除！\n");

return ;

}

while(i!=NULL){

tag=0;

for(j=i->next;j!=NULL;){

if(j->num==i->num){

i->next=j->next;

free(j);

tag=1;

break;

}else{

jp=j;

j=j->next;

}

if(jp->next==NULL){ //到末端，改变尾指针

jp=NULL;

}

}

if(tag==0)

i=i->next;

}

}