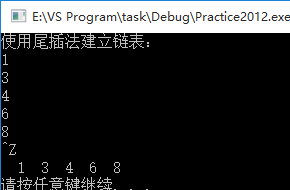
1. 有一个学生结构体，其数据成员有： 学号， 姓名， 3 门课程。从键盘上输入 5 个学生的信息。要求输出：
   1. 按照学号递增输出全部学生信息，每个学生的信息一行。（格式： 学号 姓名 分数

1 分数 2 分数 3 总分）

* 1. 输出每门课程最高分的学生的信息
  2. 输出每门课程的平均分
  3. 按照总分输出学生排名

1. 用尾插法建立链表。



void creat\_tail(pnode\*head,pnode\*tail)

{

int val;

while(fflush(stdin),scanf("%d",&val)!=EOF)

{

pnode p=(pnode)malloc(sizeof(node));

p->data=val;

p->next=NULL;

if (\*head==NULL)

{

\*head=p;

\*tail=p;

}

else

{

(\*tail)->next=p;

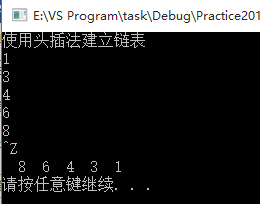
\*tail=p;

}

}

}

1. 用头插法建立链表。



void creat\_head(pnode \*head,pnode\*tail)

{

int val;

while(fflush(stdin),scanf("%d",&val)!=EOF)

{

pnode p=(pnode)malloc(sizeof(node));

p->data=val;

p->next=NULL;

if (\*head==NULL)

{

\*head=p;

\*tail=p;

//p->next=NULL;

}

else

{

p->next=\*head;

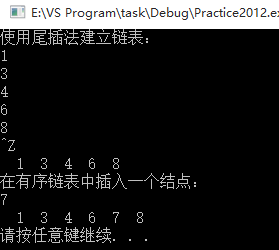
\*head=p;

}

}

}

1. 用有序插入建立链表。



void insert(pnode\*head,pnode\*tail,int val)

{

pnode pcur=NULL,pre=NULL;

pnode p=(pnode)malloc(sizeof(node));

p->data=val;

p->next=NULL;

pcur=\*head;

if (pcur==NULL)

{

\*head=p;

\*tail=p;

return;

}

if (val<(\*head)->data)

{

p->next=\*head;

\*head=p;

return ;

}

while (pcur!=NULL)

{

if (pcur->data<val)

{

pre=pcur;

pcur=pcur->next;

}else break;

}

if (pcur==NULL)

{

(\*tail)->next=p;

\*tail=p;

}

else

{

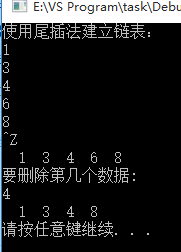
p->next=pcur;

pre->next=p;

}

}

1. 删除指定的某个结点。



void del(pnode \*head,int pos)

{

pnode pcur=\*head,pre=\*head;

int n=1;

if (pcur==NULL)

printf("链表中没有元素\n");

if(pos==1)

{

\*head=pcur->next;

free(pre);

return;

}

while (pcur!=NULL)

{

if (n!=pos)

{

pre=pcur;

pcur=pcur->next;

n++;

}

else

{

pre->next=pcur->next;

free(pcur);

return;

}

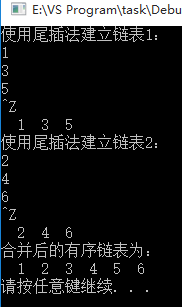
}

printf("链表中的节点数小于%d\n",pos);

return;

}

1. 将两个有序链表合并成一个有序链表。



pnode mergelist(pnode head1,pnode head2)

{

pnode head=NULL,pcur=NULL,tail=NULL;;

if(head1==NULL)

return(head=head2);

else if(head2==NULL)

return(head=head1);

head=head1->data<head2->data?head1:head2;

tail=head;

while (head1!=NULL&&head2!=NULL)

{

if (head1->data<head2->data)

{

pcur=head1;

head1=head1->next;

tail->next=pcur;

tail=pcur;

pcur->next=NULL;

}

else

{

pcur=head2;

head2=head2->next;

tail->next=pcur;

tail=pcur;

pcur->next=NULL;

}

}

if (head1==NULL)

{

tail->next=head2;

return head;

}

else

{

tail->next=head1;

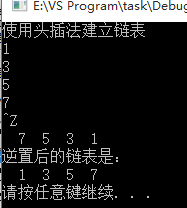
return head;

}

}

1. 将一个链表逆置。

如： 1->2 ->3 ->4 ->5 ->NULL， 输出: 5 -> 4 -> 3 ->2 ->1 -> NULL ；



void invertlist(pnode\* head,pnode\*tail)

{

pnode newhead=NULL,pre=NULL,newtail=NULL;

while ((\*head)->next!=NULL)

{

pre=\*head;

while (pre->next!=\*tail)

pre=pre->next;

pre->next=NULL;

if (newhead==NULL)

{

newhead=\*tail;

newtail=\*tail;

}

else

{

newtail->next=\*tail;

newtail=\*tail;

}

\*tail=pre;

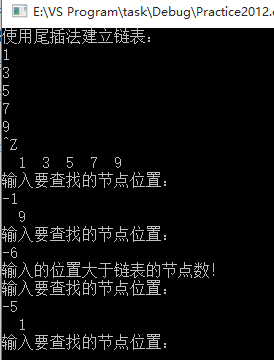
}

newtail->next=\*tail;

\*head=newhead;

}

1. 找出链表的倒数第四个节点



void findele(pnode head,int pos)// 找出链表中的元素，pos大于0表示正数第几个元素，小于0表示倒数第几个元素

{

int n=1;

pnode p=head;

if(pos>0)

{

while (head!=NULL)

{

if (pos==n)

{

printf("%3d\n",head->data);

return ;

}

head=head->next;

n++;

}

printf("输入的位置大于链表的节点数！\n");

return ;

}

else if(pos==0)

{

printf("请输入正确的位置！\n");

return ;

}

else

{

while (head!=NULL)

{

n++;

head=head ->next;

}

if (n+pos-1<0)

{

printf("输入的位置大于链表的节点数！\n");

return;

}

for (int i = 0; i < n+pos-1; i++)

{

p=p ->next;

}

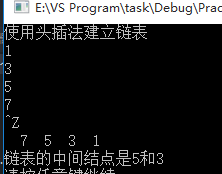
printf("%3d\n",p ->data);

return ;

}

}

1. 找出链表的中间节点



void findmid\_ele(pnode head)

{

int n=0,i;

pnode pcur=NULL;

pcur=head;

if (pcur==NULL)

{

printf("链表中没有节点\n");

return;

}

while (pcur!=NULL)

{

n++;

pcur=pcur->next;

}

if (n%2==0)

{

pcur=head;

for ( i = 1; i < n/2; i++)

pcur=pcur->next;

printf("链表的中间结点是%d和%d\n",pcur->data,pcur->next->data);

}

else

{

pcur=head;

for ( i = 0; i < n/2; i++)

pcur=pcur->next;

printf("链表的中间结点是%d\n",pcur->data);

}

}

1. 判断单链表是否有环



void isloop(pnode head)

{

pnode slow=head,fast=head;

while (fast&&fast->next)

{

slow=slow->next;

fast=fast->next->next;

if (fast==slow)

{

printf("there is a loop");

return;

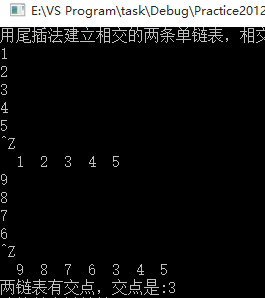
}

}

printf("there is not a loop");

}

1. 判断两个链表是否相交， 如果相交， 计算交点



void find\_two\_list\_intersection(pnode head1,pnode head2)

{

int len1=1,len2=1;//分别记录两条链表的长度

pnode tail1=NULL,tail2=NULL ;//分别指向两条链表的尾节点

pnode p=NULL,p1=NULL;

pnode pcur1=head1,pcur2=head2;

while (pcur1->next!=NULL)

{

pcur1=pcur1->next;

tail1=pcur1;

len1++;

}

while (pcur2->next!=NULL)

{

pcur2=pcur2->next;

tail2=pcur2;

len2++;

}

if (tail1==tail2)

{

pcur1=head1;

pcur2=head2;

p=len1>len2?pcur1:pcur2;

for (int i = 0; i < abs(len1-len2); i++)

{

p=p->next;

}

p1=len1>len2?pcur2:pcur1;

while (p!=p1)

{

p=p->next;

p1=p1->next;

}

printf("两链表有交点，交点是:%d\n",p1->data);

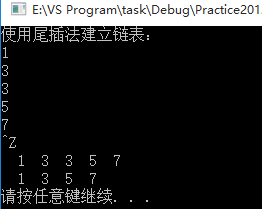
}

else

printf("两链表没有交点\n");

}

1. 删除单链表中重复的元素



void del\_same\_ele(pnode \*head)

{

pnode newhead=NULL;

pnode pcur=NULL,pre=NULL;

newhead =\*head;

\*head=NULL;

while (newhead!=NULL)

{

if (\*head==NULL)

{

\*head=newhead;

newhead=newhead->next;

(\*head)->next=NULL;

}

else

{

pcur=\*head;

while (pcur!=NULL)

{

if (pcur->data==newhead->data)

break;

pre=pcur;

pcur=pcur->next;

}

if (pcur==NULL)

{

pre->next=newhead;

pre=newhead;

newhead=newhead->next;

pre->next=NULL;

}

else

newhead=newhead->next;

}

}

print\_list(\*head);

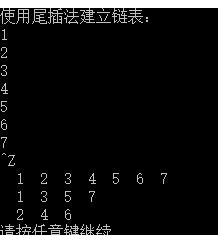
}

1. 将一个链表拆分（将链表奇数位置上的节点构成一个链表，偶数位置上的节点构成另一个链表）

例如：L：1 -> 2 -> 3 ->4 ->5 ->6 ->7 ->8 ->9 -> NULL

L1: 1 ->3 ->5 ->7 ->9 -> NULL ;

L2: 2 ->4 ->6 ->8 -> NULL



void divide\_list(pnode head)

{

int n=1;

pnode head1=NULL,head2=NULL,tail1=NULL,tail2=NULL;

while (head!=NULL)

{

if (n%2!=0)

{

if (head1==NULL)

{

head1=head;

tail1=head;

head=head->next;

tail1->next=NULL;

}

else

{

tail1->next=head;

tail1=head;

head=head->next;

tail1->next=NULL;

}

}

else

{

if (head2==NULL)

{

head2=head;

tail2=head;

head=head->next;

tail2->next=NULL;

}

else

{

tail2->next=head;

tail2=head;

head=head->next;

tail2->next=NULL;

}

}

n++;

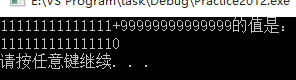
}

print\_list(head1);

print\_list(head2);

}

14．大整数加法。（计算两个整数（该整数的值可能超过一个整型变量所能表示的范围）的加法）



void big\_integer\_add(char\*str1,char\*str2)

{

pnode head1=NULL,head2=NULL,tail=NULL,p=NULL,head=NULL;

pnode pcur1=NULL,pcur2=NULL;

int flag=0,sum=0;

while (\*str1!='\0')

{

p=(pnode)malloc(sizeof(node));

p->data=\*str1-'0';

p->next=NULL;

if (head1==NULL)

{

head1=p;

}

else

{

p->next=head1;

head1=p;

}

str1++;

}

while (\*str2!='\0')

{

p=(pnode)malloc(sizeof(node));

p->data=\*str2-'0';

p->next=NULL;

if (head2==NULL)

{

head2=p;

}

else

{

p->next=head2;

head2=p;

}

str2++;

}

pcur1=head1;

pcur2=head2;

while (pcur1!=NULL||pcur2!=NULL)

{

sum=pcur1->data+pcur2->data+flag;

if( sum>=10)

{

pcur1->data=sum%10;

flag=1;

if (head==NULL)

{

head=pcur1;

head1=head1->next;

head->next=NULL;

pcur1=head1;

head2=head2->next;

free(pcur2);

pcur2=head2;

}

else

{

head1=head1->next;

pcur1->next=head;

head=pcur1;

pcur1=head1;

head2=head2->next;

free(pcur2);

pcur2=head2;

}

}

else

{

pcur1->data=sum;

flag=0;

if (head==NULL)

{

head=pcur1;

head1=head1->next;

head->next=NULL;

pcur1=head1;

head2=head2->next;

free(pcur2);

pcur2=head2;

}

else

{

head1=head1->next;

pcur1->next=head;

head=pcur1;

pcur1=head1;

head2=head2->next;

free(pcur2);

pcur2=head2;

}

}

}

if (head1!=NULL)

{

while (head1!=NULL)

{

sum=pcur1->data+flag;

if (sum>=10)

{

pcur1->data=sum%10;

flag=1;

if (head==NULL)

{

head=pcur1;

head1=head1->next;

head->next=NULL;

pcur1=head1;

}

else

{

head1=head1->next;

pcur1->next=head;

head=pcur1;

pcur1=head1;

}

}

else

{

pcur1->data=sum;

flag=0;

if (head==NULL)

{

head=pcur1;

head1=head1->next;

head->next=NULL;

pcur1=head1;

}

else

{

head1=head1->next;

pcur1->next=head;

head=pcur1;

pcur1=head1;

}

}

}

}

if (head2!=NULL)

{

while (head2!=NULL)

{

sum=pcur2->data+flag;

if (sum>=10)

{

pcur2->data=sum%10;

flag=1;

if (head==NULL)

{

head=pcur2;

head2=head2->next;

head->next=NULL;

pcur2=head2;

}

else

{

head2=head2->next;

pcur2->next=head;

head=pcur2;

pcur2=head2;

}

}

else

{

pcur2->data=sum;

flag=0;

if (head==NULL)

{

head=pcur2;

head2=head2->next;

head->next=NULL;

pcur2=head2;

}

else

{

head2=head2->next;

pcur2->next=head;

head=pcur2;

pcur2=head2;

}

}

}

}

if (flag==1)

{

pcur1=(pnode)malloc(sizeof(node));

pcur1->data=1;

pcur1->next=head;

head=pcur1;

}

print\_list(head);

}