## 实验一 线性表的操作

## **1.线性表的基本操作**

## **1.1 需求分析**

A、顺序表的输入形式如下：

（1）顺序表的定位、取值

输入顺序表

定位

数值

位置

-1退出

取值

位置

数值

0/超出数组大小退出

（2）顺序表的插入、删除

随机数组

Insert

插入数值 插入位置

插入后数组

delete

删除数据

删除后的数组

（3）顺序表的合并

输入数组1数组2大小

输出随机数组1

输出随机数组2

输出合并的数组

功能：利用顺序表实现线性表的定位、取值、插入、删除和合并

数据范围：1~100

测试数据：随机数 正确执行各项操作

B、链表的输入形式如下：

（1）链表的定位

输入单链表的各个数据，输入-1结束

输入定位的数值

输出数值位置

1. 链表的取值

输入单链表的各个数据，输入-1结束

输入位置

输出该位置的数值

1. 链表的插入

输入一个单链表，输入-1结束

输入插入的数值、位置

输出插入后的单链表

1. 链表的删除

输入一个单链表，输入-1结束

输入需要删除的数据

输出删除后的单链表

1. 链表的合并

输入按值非递减单链表A，输入-1结束

输入按值非递减单链表B，输入-2结束

输出插入后的按值非递减单链表

功能：利用链表实现线性表的定位、取值、插入、删除和合并

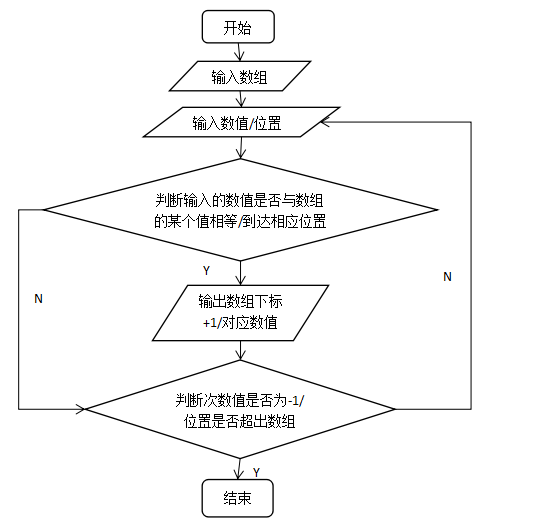
数据范围：1~100

测试数据：正确输入：1~100 正确执行 错误输入：数据超出范围的数据 跳出程序

**1.2 概要设计**

A、顺序表：

（1）顺序表的定位、取值



1. 顺序表的插入、删除

生成随机数组→输入插入数值、插入位置→插入位置后面数值往后摞，将插入数值放入插入位置→输出插入后数组→输入删除数据→将该数值的数组后面的数值全部往前放一个→输出删除后的数组

1. 顺序表的合并

生成随机数组→建立第三个数组，长度为数组1和数组2之和→将数组2复制到数组1后面

B、链表：

（1）链表的定位

建立单链表→调用定位函数→输出数值

1. 链表的取值

建立单链表→调用取值函数→输出位置

1. 链表的插入

建立单链表→调用插入函数→输出插入后的单链表

1. 链表的删除

建立单链表→调用删除函数→输出删除后的单链表

1. 链表的合并

建立按值非递减单链表AB→调用合并函数→输出合并后的函数

**1.3 详细设计**

A、顺序表：

（1）顺序表的定位、取值

创建list[size]

cin>>m

for循坏判断list[i]==m

if list[i]==m ,cout<<i

cin>>number

for循坏到list[number], cout<<list[number]

1. 顺序表的插入删除

随机生成数组

输入插入数值、位置

for循环到相应位置，将数组的数值后摞一个位置

将插入数值放入插入位置

输出插入后的数组

输入删除数组

for循环删除该数值，所有后续数值前摞一个位置

输出删除后的数组

1. 顺序表的合并

输入数组1，数组2大小

随机数组1

随机数组2

将数组2复制到数组1后面并输出合并后的数组1

B、链表：

（1）链表的定位

建立单链表

cin>>number

int k=1;

while(L->next!=NULL)

{

If（结点数据==number）//判断该结点的数据与定位数值是否相等

cout<<k;

else

{

1. >next=L->next->next; //下一个结点

k++;

}

}

1. 链表的取值

建立单链表

输入取值的位置

循环到相应的位置

输出该位置的数值

1. 链表的插入

建立单链表

输入插入的数值、位置

循坏判断下一个结点的数据是否是插入位置

If（是）

{

新建结点，赋值，指针下一个

此结点的next指向新建结点

}

else

{

结点指向下一个

位置+1

}

1. 链表的删除

建立单链表

输入需要删除的数据

循坏判断下一个结点的数值是否为该数据

If（是）

{

新建结点

将下一结点的数据、指针赋给新建结点

此结点指向下一个的下一个

释放新建结点

}

else

{

此结点指向下一个结点

}

1. 链表的合并

建立按值非递减单链表AB

判断AB结点的数值大小

新建单链表，新建结点

将小的数值赋给新建结点

新建结点是单链表的下一个结点

**1.4 调试分析**

A、顺序表：

算法时间复杂度为n，空间复杂度根据数组大小而定，可以建立动态数组，要求用户输入数组大小。

B、链表：

时间复杂度n，空间复杂度取决输入数值数量，操作时容易丢失指针，特别是操作第一个数据时，可以新建结点替代需要调用的结点。定位取值等操作单独实现较易，一起实现容易造成指针位置不明。

**1.5 用户使用说明**

A、顺序表：

（1）顺序表的定位、取值

当程序输出定位时，用户输入显示数据中的一个，按下回车键，程序会输出该数据所在位置，输入-1跳出循坏；程序输出取值时，用户输入位置，按下回车键，程序会输出该位置的数据，没有该位置时跳出程序。

1. 顺序表的插入、删除

程序生成随机数组，提示Insert时，先输入插入的数据，空格输入插入位置，回车输出插入后的数组，提示delete时，输入想删除的数据，回车输出删除后的数组。

1. 顺序表的合并

输入数组1和数组2大小即可

B、链表：

（1）链表的定位

输入新建链表的数据，遇到-1时结束，提示定位时，输入定位数据，按回车键，输出该位置的数据。

1. 链表的取值

输入新建链表的数据，遇到-1时结束，提示取值时，输入位置，按回车键，输出该位置的数据。

1. 链表的插入

输入新建链表的数据，遇到-1时结束，提示插入时，输入插入的数据、位置，按回车键，输出插入后后的单链表。

（4）链表的删除

输入新建链表的数据，遇到-1时结束，提示删除时，输入删除数据，按回车键，输出删除后的单链表。

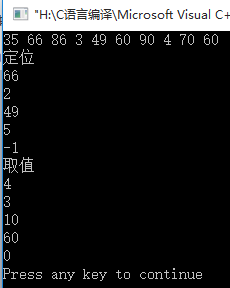
1. 链表的合并

输入按值非递减链表A的数据，遇到-1时结束，输入按值非递减单链表B的数据，遇到-2时结束，按回车键，输出合并后的按值非递减后的单链表。

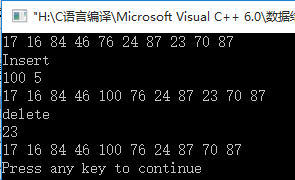
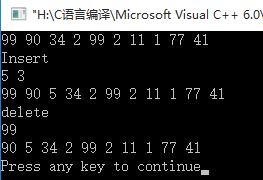
**1.6 测试结果**

A、顺序表：

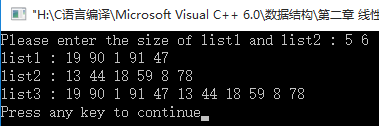
（1）顺序表的定位、取值

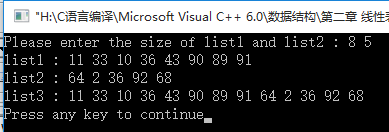
****

1. 顺序表的插入、删除



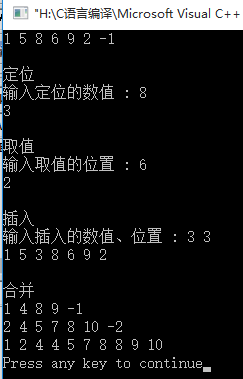
1. 顺序表的合并



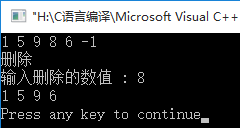


B、链表：

（1）链表的定位、取值、插入、合并



1. 链表的删除



**1.7 附录**

A、顺序表：

（1）顺序表的定位、取值

#include<iostream>

#include<stdlib.h>

#include<time.h>

using namespace std;

int main()

{

srand(time(0));

int list[10];

for (int i=0;i<10;i++) //建立随机数组

{

list[i]=rand()%100+1;

cout<<list[i]<<" ";

}

cout<<endl<<"定位"<<endl;

int k;

cin>>k;

while(k!=-1) //循坏判定输入数值的位置

{

for (int j=0;j<10;j++)

if (list[j]==k)

cout<<j+1<<endl;

cin>>k;

}

cout<<"取值"<<endl; //循坏判定输入位置的数值

int n;

cin>>n;

while (n>0&&n<=10)

{

cout<<list[n-1]<<endl;

cin>>n;

}

return 0;

}

1. 顺序表插入与删除

int insert(int list1[],int num,int position)

{

if (position<0||position>10) //输入插入位置超出数组的数据范围，退出程序

return -1;

for (int i=9;i>=position-1;i--) //将插入位置及后面的数据全部后摞一个

list1[i+1]=list1[i];

list1[position-1]=num; //将插入数据赋给插入位置

for (int j=0;j<11;j++) //输出插入后的数组

cout<<list1[j]<<" ";

return 0;

}

int deletenum(int list2[],int number)

{

int m;

for (int i=0;i<11;i++) //循环寻找删除数据的位置

{

if (number==list2[i])

{

m=i;break;

}

else if(number!=list2[i]&&i==10) //超出数组范围，退出程序

return -1;

}

for (int j=m;j<10;j++) //将删除位置后面的数值全部往前摞一位

list2[j]=list2[j+1];

for (int k=0;k<10;k++) //输出删除后的数组

cout<<list2[k]<<" ";

cout<<endl;

return 0;

}

1. 顺序表的合并

int merge(int list1[],int list2[],int a,int b);

int main()

{

srand(time(0));

int list1[100],list2[100];

int a, b;

cout<<"Please enter the size of list1 and list2 : ";

cin>>a>>b;

cout<<"list1 : "; //创建顺序表1

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

{

list1[i]=rand()%100+1;

cout<<list1[i]<<" ";

}

cout<<endl<<"list2 : "; //创建顺序表2

for (int j=0;j<b;j++)

{

list2[j]=rand()%100+1;

cout<<list2[j]<<" ";

}

cout<<endl<<"list3 : ";

merge(list1,list2,a,b);

return 0;

}

int merge(int list1[],int list2[],int a,int b)

{

for (int i=a;i<a+b;i++) //将顺序表2加到顺序表1后面

list1[i]=list2[i-a];

for (int j=0;j<a+b;j++) //输出合并后的顺序表1

cout<<list1[j]<<" ";

cout<<endl;

return 0;

}

1. 链表
2. 链表的定位、取值、插入、合并

#include<iostream>

using namespace std;

typedef struct node{ //建立结构体

int date;

struct node \*next;

}node;

void print(node \*L); //打印链表

int locate(node \*L,int number); //链表定位

int evaluation(node \*L,int i); //链表的取值

void insert(node \*L,int number,int i); //链表的插入

//void deleten(node \*L,int number); //链表的删除

void merge(node \*L1,node \*L2); //链表的合并

int main()

{

node \*p, \*r, \*L;

int number, i;

L=new node;

r=L;

cin>>number;

while (number!=-1) //新建链表L

{

p=new node;

p->date=number;

r->next=p;

r=p;

cin>>number;

}

r->next=NULL;

cout<<endl<<"定位"<<endl<<"输入定位的数值 : ";

cin>>number;

locate(L,number); //链表定位

cout<<endl<<"取值"<<endl<<"输入取值的位置 : ";

cin>>i;

evaluation(L,i); //链表的取值

cout<<endl<<endl<<"插入"<<endl<<"输入插入的数值、位置 : ";

cin>>number>>i;

insert(L,number,i); //链表的插入

cout<<endl<<"合并"<<endl; //链表合并

node \*L1, \*L2, \*p1, \*p2, \*r1, \*r2;

int number1, number2;

L1=new node;

r1=L1;

L2=new node;

r2=L2;

cin>>number1;

while (number1!=-1) //新建链表1

{

p1=new node;

p1->date=number1;

r1->next=p1;

r1=p1;

cin>>number1;

}

r1->next=NULL;

cin>>number2;

while (number2!=-2) //新建链表2

{

p2=new node;

p2->date=number2;

r2->next=p2;

r2=p2;

cin>>number2;

}

r2->next=NULL;

merge(L1,L2); //链表1、2的合并

return 0;

}

void print(node \*L) //打印链表

{

while (L->next!=NULL) //下一个结点为空时跳出

{

cout<<L->next->date<<" "; //输出此结点的数据

L->next=L->next->next; //转到下一个结点

}

cout<<endl;

}

int locate(node \*L,int number) //链表定位

{

node \*r;

r=L->next;

int i=0;

while (r!=NULL) //结点为空时跳出循坏

{

if ((r->date)==number) //判断此结点的数据是否为定位数据

{

cout<<++i<<endl;

break;

}

else

{

++i; //位置加1

r=r->next; //指针指向下一个结点

}

}

return i;

}

int evaluation(node \*L,int i) //链表的取值

{

node \*r;

r=L->next;

int j=1;

while (r!=NULL) //结点为空时跳出循坏

{

if (j==i)

{

cout<<r->date; //输出该位置的数据

break;

}

else

{

++j; //位置+1

r=r->next; //指针指向下一个结点

}

}

return 0;

}

void insert(node \*L,int number,int i) //链表的插入

{

node \*p, \*r, \*t;

if(i==1) //如果插入位置为第一个时

{

p=new node;

p->date=number; //将数据赋给新建结点的数据域

p->next=L->next; //头指针指向下一个的下一个

p=L; //将头指针赋给p

}

else

{

r=L->next;

int k=1;

while (r!=NULL) //结点为空时跳出循坏

{

if (k+1==i) //如果下一个结点是插入位置

{

p=new node; //新建结点

p->date=number; //数值赋给数据域

p->next=r->next; //指针指向下一个的下一个

r->next=p; //指针指向新结点

r=p; //将新结点赋给此结点

break; //跳出循坏

}

else

{

r=r->next; //指针指向下一个

++k; //位置+1

}

}

}

print(L); //输出插入后的链表

}

void merge(node \*L1,node \*L2) //链表的合并

{

node \*p, \*r1, \*r2, \*L3, \*p3, \*r3;

L3=new node;

r3=L3;

r1=L1;

r2=L2;

while (r1->next!=NULL&&r2->next!=NULL) //两个链表中的一个结点为空时跳出循坏

{

if (r1->next->date<r2->next->date) //链表1的数据小于链表2的数据

{

p=new node; //新建结点

p->date=r1->next->date; //将链表1的数据赋给新建结点的数据域

r3->next=p; //链表3的指针指向新建结点

r3=p; //将新建结点赋给链表3的结点

r1->next=r1->next->next; //链表1的指针指向下一个结点的指针域

}

else if (r1->next->date==r2->next->date) //链表1的数据与链表2的数据

{

p=new node; //新建结点

p->date=r1->next->date; //将链表1的的数据赋给新结点

r3->next=p; //链表3指向新建结点

r3=p; //将新建结点赋给链表3的结点

p=new node; //同理，将链表2的此结点拷贝给链表3的下一个结点

p->date=r2->next->date;

r3->next=p;

r3=p;

r1->next=r1->next->next; //链表1的指针指向下一个结点

r2->next=r2->next->next; //链表2的指针指向下一个结点

}

else if (r1->next->date>r2->next->date) //链表1的数据大于链表2的数据

{

p=new node; //新建结点

p->date=r2->next->date; //将链表2的数据赋给此结点

r3->next=p; //链表3指向新建结点

r3=p; //将新建结点赋给链表3

r2->next=r2->next->next; //链表2的指针指向下一个结点

}

}

while (r1->next!=NULL) //链表1不为空

{

p=new node; //建立新的结点

p->date=r1->next->date; //将链表1的数据赋给新建结点

r3->next=p; //将链表3指向下一个结点

r3=p; //将新建结点赋给链表3的此结点

r1->next=r1->next->next; //链表1的指针指向下一个结点

}

while (r2->next!=NULL) //链表2不为空时

{

p=new node; //新建结点

p->date=r2->next->date; //将链表2的数据赋给新建结点

r3->next=p; //链表3指向新建结点

r3=p; //将新建结点赋给链表3的此结点

r2->next=r2->next->next; //链表2的指针指向下一个结点

}

r3->next=NULL; //链表3的末结点指向空

print(L3); //输出链表3

}

1. 链表的删除

#include <iostream>

using namespace std;

typedef struct node{ //构建链表结构体

int date;

struct node \*next;

}node;

int main()

{

node \*p, \*r, \*L;

int number;

L=new node;

r=L;

cin>>number; //新建链表

while (number!=-1)

{

p=new node;

p->date=number;

r->next=p;

r=p;

cin>>number;

}

r->next=NULL;

cout<<"删除"<<endl<<"输入删除的数值 : ";

cin>>number;

if((L->next->date)==number) //删除数值为链表的第一个结点

{

p=new node; //新建结点

p->date=L->next->date; //链表的首结点的数据赋给新建结点的数据

p->next=L->next->next; //新建结点的指针指向链表的下一个指针域

L->next=L->next->next; //链表的指针指向下一个结点的指针

delete p; //删除新建结点

}

else

{

r=L->next; //将链表的首结点赋给指针r

while (r!=NULL) //指针为空时跳出循坏

{

if ((r->next->date)==number) //下一个结点的数据等于删除数据

{

p=new node; //新建结点

p->date=r->next->date; //将链表的下一个结点的数据赋给新建结点

p->next=r->next->next; //新建结点指向链表的下一个结点的下一个指针

r->next=r->next->next; //链表的此结点的指针指向下一个结点的下一个

delete p; //删除新建结点

break; //跳出循坏

}

else

{

r=r->next; //链表指向下一个结点的指针

}

}

}

while(L->next!=NULL) //指针为空时跳出循坏

{

cout<<L->next->date<<" "; //输入此结点的数据

L->next=L->next->next; //指针指向下一个指针域

}

cout<<endl;

return 0;

}

1. **一元n次加法操作**
2. **需求分析**
3. 顺序表

输出形式：

array1(column\_size1=5) : (the first row is exponent , the second row is coefficient .)

指数行

系数行

array2(column\_size2=8) : (the first row is exponent , the second row is coefficient .)

指数行

系数行

array3 : (the first row is exponent , the second row is coefficient .)

相加后的指数行

相加后的系数行

功能：用户输入两个一元多项式，程序实现相加，输出最后结果

数据范围：

指数：正整数 系数：全体实数

测试数据：

正确输入：输入值范围内 正确结果 错误输入：输入值超出规定范围 跳出程序

1. 链表

输出形式：

L1 :

coefficient exponent (exp=-1 to quit)

系数 指数

L2 :

coefficient exponent (exp=-2 to quit)

系数 指数

L3 :

coefficient exponent

相加后的系数 相加后的指数

功能：用户输入两个一元多项式，程序实现相加，输出最后结果

数据范围：

指数：正整数 系数：全体实数

测试数据：

正确输入：输入值范围内 正确结果 错误输入：输入值超出规定范围 跳出程序

**2．概要设计**

A、顺序表

开始

创建二维数组1、2、3，初始化数组3全为零

输入二维数组1、2的各项系数指数

二维数组1、2是否已完全输入

Y

N

比较每一项的指数是否相等

Y

系数相加是否为零

N

Y

N

数组1的指数>数组2的指数

数组3的指数=数组1的指数，数组3的系数=数组1的系数+数组2的系数

数组1、2下摞一位

N

Y

数组3的系数指数=数组2的系数指数

数组3的系数指数=数组1的系数指数

数组1、2、3下摞一位

数组2、3下摞一位

数组1、3下摞一位

数组1是否完全输出

数组2的所有剩余项赋给数组3

Y

N

输出数组3指数不为零的所有项

数组1的所有剩余项赋给数组3

结束

1. 链表

开始

创建链表1、2、3

输入链表1、2的各项系数指数

链表1、2的指针否为空

Y

N

比较每一项的指数是否相等

系数相加是否为零

Y

Y

N

N

链表1的指数>链表2的指数

链表3的指数=链表1的指数，链表3的系数链表1的系数+链表2的系数

链表1、2指针指向下一个结点

N

Y

链表3的系数指数=链表2的系数指数

链表3的系数指数=链表1的系数指数

链表1、2、3指针指向下一个结点

链表2、3指针指向下一个结点

链表1、3指针指向下一个结点

链表1是否完全为空

链表2的所有剩余项赋给链表3

Y

N

输出链表组3指数不为零的所有项

链表1的所有剩余项赋给链表3

结束

**3．详细设计**

A、顺序表

创建二维数组1、2、3

初始化数组3每项均为0

输入数组1、2的每一项系数、指数

while(数组1、2有一个输出完全）

{

比较数组1、2的指数大小

if(数组1指数>数组2指数）

{

数组3的系数、指数=数组2的系数、指数

数组2、3往后摞一位

}

else if (数组1的指数==数组2的指数）

{

If(数组1的系数+数组2的系数==0）

{

数组1、2往后摞一位

}

else

{

数组3的指数=数组1的指数

数组3的系数=数组1的系数+数组2的系数

数组1、2、3均往后摞一位

}

}

else if（数组1的指数<数组2的指数）

{

数组3的指数、系数=数组1的指数、系数

数组1、3往后摞一位

}

while(数组1未完全输出）

{

数组3的指数、系数=数组1的指数、系数

数组1、3往后摞一位

}

while(数组2未完全输出）

{

数组3的指数、系数=数组2的指数、系数

数组1、2往后摞一位

}

输出数组3的指数非零项

}

B、链表

创建链表1、2、3

输入链表1、2的每一项系数、指数

while(链表1、2有一个指针为空）

{

比较链表1、2的指数大小

if(链表1指数>链表2指数）

{

链表3的系数、指数=链表2的系数、指数

链表2、3指针指向下一个结点

}

else if (链表1的指数==链表2的指数）

{

If(链表1的系数+链表2的系数==0）

{

链表1、2指针指向下一个结点

}

else

{

链表3的指数=链表1的指数

链表3的系数=链表1的系数+链表2的系数

链表1、2、3指针指向下一个结点

}

}

else if（链表1的指数<链表2的指数）

{

链表3的指数、系数=链表1的指数、系数

链表1、3指针指向下一个结点

}

while(链表1的指针不为空）

{

链表3的指数、系数=链表1的指数、系数

链表1、3指针指向下一个结点

}

while(链表2的指针不为空）

{

链表3的指数、系数=链表2的指数、系数

链表1、2指针指向下一个结点

}

输出链表3

}

**4.调试分析**

A、顺序表

算法时间复杂度n2，空间复杂度由输入数组大小而定。

遇到问题：二维数组行下标错误，应该从0开始，误写为1开始；

改进：生成动态数组，由用户确定数组大小

B、链表

算法时间复杂度n，空间复杂度大小由用户输入数据多少确定。

遇到问题：头指针弄丢了，建立新的指针，将头指针赋给新指针，调用时用新指针，输出时再调用头指针。

改进：将合并的链表归于已经建立的链表1或2中，而不是新建链表3

1. **用户使用说明**
2. 顺序表

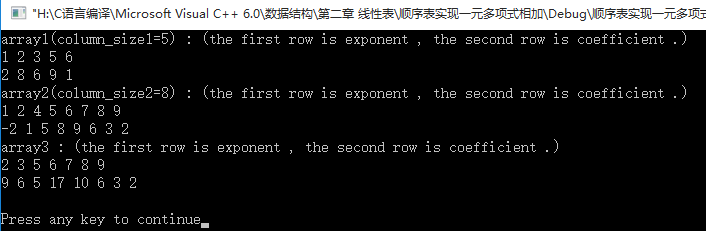
根据提示，在第一行输入多项式的指数，每输入一个，空一格，输入足够数值时，enter换行，同样的输入方法输入对应指数的系数，enter进入第二个多项式的输入，同样的输入方法，输入完成后enter键输出结果

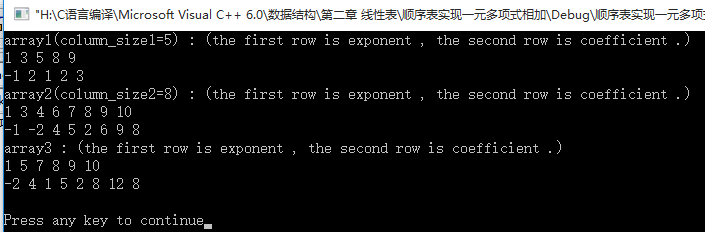
1. 链表

根据提示，先陆续输入多项式1每一项的系数、指数，先输入系数，空格，输入指数，enter换行，输入下一项，完成输入后，下一项的系数随意，指数输入为-1,跳出输入多项式1，接下来，同样输入方法输入多项式2，输入完成后，最后一项的系数随意，指数输入-2。

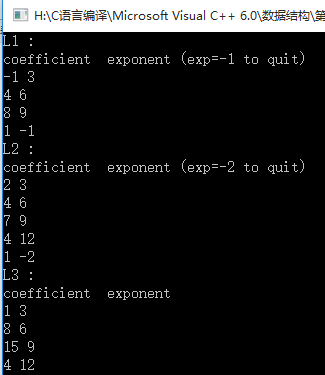
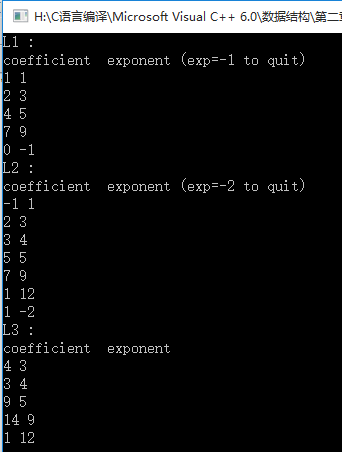
1. **测试结果**

A、顺序表





1. 链表



1. **附录**

A、顺序表

#include<iostream>

using namespace std;

//第一行指数，第二行系数

const int column\_size1=5; //二维数组1的列数

const int column\_size2=8; //二位数组2的列数

const int column\_size3=13; //二维数组3的列数

//合并二维数组12函数声明

int merge(const int array1[2][column\_size1],const int array2[2][column\_size2]);

int main()

{

int array1[2][column\_size1], array2[2][column\_size2]; //声明二维数组12

//创建array1

cout<<"array1(column\_size1=5) : (the first row is exponent , the second row is coefficient .) "<<endl;

for (int row1=0; row1<2; row1++) //循环输入二维数组1的数据，此循环控制行

{

for (int col1=0; col1<column\_size1; col1++) //循环控制每行列的每个数据的输入

cin>>array1[row1][col1]; //输入数据

}

//创建array2

cout<<"array2(column\_size2=8) : (the first row is exponent , the second row is coefficient .) "<<endl;

for (int row2=0; row2<2; row2++) //二维数组2的行控制

{

for (int col2=0; col2<column\_size2; col2++) //二位数组2的列控制

cin>>array2[row2][col2]; //输入每行每列的数据

}

merge(array1, array2); //调用合并函数，合并二维数组12

return 0;

}

int merge(const int array1[2][column\_size1],const int array2[2][column\_size2])

{

int array3[2][column\_size3]; //声明二维数组3

//创建array3

for (int row3=0; row3<2; row3++) //控制二维数组3的行

{

for(int col3=0; col3<column\_size3; col3++) //控制二维数组3的列

array3[row3][col3]=0; //初始化每个数据均为0

}

int a, b, c;

a=b=c=0;

while(a<column\_size1&&b<column\_size2)

//a控制二维数组1的列数，b控制二维数组2的列数

{

if (array1[0][a]==array2[0][b]) //二维数组12的数据相等

{

if (array1[1][a]+array2[1][b]==0) //系数相加为0

{

++a; //二维数组1的列数+1

++b; //二维数组2的列数+1

}

else //系数相加不为0

{

array3[0][c]=array1[0][a];

//系数赋值，二维数组3的第一行此列的数据等于二维数组1的此列的数据

array3[1][c]=array1[1][a]+array2[1][b];

//指数赋值，二维数组3的第二行此列的数据等于二维数组1和二维数组2此列的数据之和

++a; //二维数组1的列数+1

++b; //二维数组2的列数+1

++c; //二位数组3的列数+1

}

}

else if (array1[0][a]<array2[0][b]) //二维数组1的指数小于二维数组2的指数

{

array3[0][c]=array1[0][a];

//数组3的第一行的此列数据=数组1的第一行的此列数据

array3[1][c]=array1[1][a];

//数组3的第二行的此列数据=数组1的第二行的此列数据

++c; //数组3的列数+1

++a; //数组1的列数+1

}

else if (array1[0][a]>array2[0][b]) //数组1的指数>数组2的指数

{

array3[0][a]=array2[0][b];

//数组3的第一行的此列数据=数组2的第一行的此列数据

array3[1][c]=array2[1][b];

//数组3的第二行的此列数据=数组2的第二行的此列数据

++c; //数组3的列数+1

++b; //数组2的列数+1

}

}

if (a<column\_size1) //数组1未输出完毕

{

for (; a<column\_size1; a++) //控制数组1的列数

{

array3[0][c]=array1[0][a];

//数组3的第一行的此列数据=数组1的第一行的此列数据

array3[1][c]=array1[1][a];

//数组3的第二行的此列数据=数组1的第二行的此列数据

++c; //数组3的列数+1

}

}

if (b<column\_size2) //数组2未输出完毕

{

for (; b<column\_size2; b++)

{

array3[0][c]=array2[0][b];

//数组3的第一行的此列数据=数组2的第一行的此列数据

array3[1][c]=array2[1][b];

//数组3的第二行的此列数据=数组2的第二行的此列数据

++c; //数组3的列数+1

}

}

//输出array3

cout<<"array3 : (the first row is exponent , the second row is coefficient .) "<<endl;

for (int j=0; j<2; j++) //控制数组3的行数

{

for (int col=0; col<column\_size3; col++) //控制数组3的列数

{

if(array3[j][col]!=0) //如果数组3的数据不为0

cout<<array3[j][col]<<" "; //输出此项数据

}

cout<<endl;

}

cout<<endl;

return 0;

}

B、链表

#include<iostream>

#include<stdlib.h>

#include<time.h>

using namespace std;

typedef struct node{ //创建链表的结构体

int coe;

int exp;

struct node \*next;

}node;

//AB单链表相加为C 主函数

int merge(node \*L1, node \*L2);

int main()

{

node \*L1, \*L2, \*p1, \*p2, \*t1, \*t2;

L1=new node;

t1=L1;

L2=new node;

t2=L2;

int coe1, coe2, exp1, exp2;

cout<<"L1 : "<<endl<<"coefficient exponent (exp=-1 to quit)"<<endl;

cin>>coe1>>exp1;

while(exp1!=-1) //输入链表1的多项式的每一项指数，系数

{

p1=new node;

p1->coe=coe1;

p1->exp=exp1;

t1->next=p1;

t1=p1;

cin>>coe1>>exp1;

}

t1->next=NULL;

cout<<"L2 : "<<endl<<"coefficient exponent (exp=-2 to quit)"<<endl;

cin>>coe2>>exp2;

while (exp2!=-2) //输入链表1的多项式的每一项指数，系数

{

p2=new node;

p2->coe=coe2;

p2->exp=exp2;

t2->next=p2;

t2=p2;

cin>>coe2>>exp2;

}

t2->next=NULL;

merge(L1,L2); //调用合并函数

return 0;

}

//AB单链表相加为C

int merge(node \*L1, node \*L2)

{

node \*L3, \*p3, \*t3;

L3=new node;

t3=L3;

while (L1->next!=NULL&&L2->next!=NULL) //链表1、2的指针为空时跳出循坏

{

if (L1->next->exp==L2->next->exp) //链表1、2的数据相等

{

if (L1->next->coe+L2->next->coe==0) //链表1、2的数据相加为0

{

L1->next=L1->next->next; //链表的指针指向下一个结点的指针域

L2->next=L2->next->next; //链表的指针指向下一个结点的指针域

}

Else //链表1、2的数据相加不为0

{

p3=new node;

p3->coe=L1->next->coe+L2->next->coe;

//新建结点的系数等于链表1、2此结点数据的和

p3->exp=L1->next->exp;

//新建结点的指数等于链表1的指数

t3->next=p3; //链表3指向新建结点

t3=p3; //将新建结点赋给链表3的此结点

L1->next=L1->next->next; //链表1的指针指向下一个结点

L2->next=L2->next->next; //链表2的指针指向下一个结点

}

}

else if (L1->next->exp<L2->next->exp) //链表1的指数小于链表2的指数

{

p3=new node; //新建结点

p3->coe=L1->next->coe; //将链表1的系数赋给新建结点的系数

p3->exp=L1->next->exp; //将链表1的指数赋给新建结点的系数

t3->next=p3; //链表3指向新建结点

t3=p3; //将新建结点赋给链表3的此结点

L1->next=L1->next->next; //链表1的指针指向下一个结点的指针

}

else if(L1->next->exp>L2->next->exp) //链表1的指数>链表2的指数

{

p3=new node; //新建结点

p3->coe=L2->next->coe; //将链表2的系数赋给新建结点的系数

p3->exp=L2->next->exp; //将链表2的指数赋给新建结点的指数

t3->next=p3; //链表3的指针指向新建结点

t3=p3; //将新建结点赋给链表3的此结点

L2->next=L2->next->next; //链表2的指针指向下一个结点的指针域

}

}

while (L1->next!=NULL) //链表1未完全输出

{

p3=new node;

p3->coe=L1->next->coe;

p3->exp=L1->next->exp;

t3->next=p3;

t3=p3;

L1->next=L1->next->next;

}

while (L2->next!=NULL) //链表2未完全输出

{

p3=new node; //新建结点

p3->coe=L2->next->coe; //将链表2的系数赋给新建结点的系数

p3->exp=L2->next->exp; //将链表2的指数赋给新建结点的指数

t3->next=p3; //将新建结点赋给链表3的此结点

t3=p3; //将新建结点赋给链表3的此结点

L2->next=L2->next->next; //链表2的指针指向下一个结点的指针域

}

t3->next=NULL; //链表3的下一个指针域为空

cout<<"L3 : "<<endl<<"coefficient exponent"<<endl;

while (L3->next!=NULL) //链表3指针为空时跳出循环

{

cout<<L3->next->coe<<" "<<L3->next->exp<<endl;

//输出链表3的每一项的系数、指数

L3->next=L3->next->next; //链表3的指针指向下一个结点的指针

}

return 0;

}