高精度计算

本词条缺少名片图，补充相关内容使词条更完整，还能快速升级，赶紧来编辑吧！

高精度运算，是指参与运算的数(加数，减数，因子……）范围大大超出了标准数据类型（整型，实型）能表示的范围的运算。例如，求两个200位的数的和。这时，就要用到高精度算法了。

中文名	高精度计算	外文名	high precision computation

目录

定义

高精度加法

高精度运算主要解决以下三个问题：

一、加数、减数、运算结果的输入和存储

运算因子超出了整型、实型能表示的范围，肯定不能直接用一个数的形式来表示。在Pascal中，能表示多个数的数据类型有两种：数组和字符串。

数组：每个数组元素存储1位（在优化时，这里是一个重点！），有多少位就需要多少个数组元素；用数组表示数的优点：每一位都是数的形式，可以直接加减；运算时非常方便。用数组表示数的缺点：数组不能直接输入；输入时每两位数之间必须有分隔符，不符合数值的输入习惯；

字符串：String型字符串的最大长度是255，可以表示255位。Ansistring型字符串长度不受限制。用字符串表示数的优点：能直接输入输出，输入时，每两位数之间不必分隔符，符合数值的输入习惯；用字符串表示数的缺点：字符串中的每一位是一个字符，不能直接进行运算，必须先将它转化为数值再进行运算；运算时非常不方便；

综合以上所述，对上面两种数据结构取长补短：用字符串读入数据，用数组存储数据：

```
var st:string;
x,y:array[0..255]of integer;{定义两个数组,X和Y,用来储存数}
i,j,l1,l2:integer;
begin
readln(st);
l1:=length(st);{------length(x),该函数是获取字符串X的长度,返回为整型}
for i:=0 to 255 do x[i]:=0;{数组初始化，该句等价于'fillchar（x,sizeof(x),o）；'，即给一数组整体赋值，但运行速度快于用'for'语句对数组中的每一个数赋值}
for i:=l1 downto 1 do
x[l1-i+1]:=ord(st[i])-ord('0');{------这里是重点,把字符串转换为数值,储存在数组中}
readln(st);
```

```
l2:=length(st);{------length(x),该函数是获取字符串X的长度,返回为整型}

for i:=0 to 255 do y[i]:=0;{数组初始化，该句等价于'fillchar（y,sizeof(y),o）；'}

for i:=l2 downto 1 do

y[l2-i+1]:=ord(st[i])-ord('0');{------这里是重点,把字符串转换为数值,储存在数组中}
```

对字符串转为数值原理补充:ord(x)-48,如果X='1',因为'1'的ASCLL码是49,所以减去48就等于1,间接地把字符转换为数值了,各位初手要好好体会.

二、运算过程

在往下看之前，大家先列竖式计算35+86。

注意的问题：

（1）运算顺序：两个数靠右对齐；从低位向高位运算；先计算低位再计算高位；

（2）运算规则：同一位的两个数相加再加上从低位来的进位，成为该位的和；这个和去掉向高位的进位就成为该位的值；如上例：3+8+1=12，向前一位进1，本位的值是2；可借助MOD、DIV运算完成这一步；

（3）最后一位的进位：如果完成两个数的相加后，进位位值不为0，则应添加一位；

（4）如果两个加数位数不一样多，则按位数多的一个进行计算；

```
if l1<l2 then l1:=l2;

for i:=1 to l1 do

begin

x[i]:=x[i]+y[i];

x[i+1]:=x[i+1]+x[i] div 10;

x[i]:=x[i] mod 10;

end;
```

三、结果的输出（这也是优化的一个重点）

按运算结果的实际位数输出

```
var st:string;

x,y:array[0..255]of integer;

i,j,l1,l2:integer;

begin

readln(st);

l1:=length(st);

for i:=0 to 255 do x[i]:=0;

for i:=l1 downto 1 do

x[l1-i+1]:=ord(st[i])-ord('0');

readln(st);

l2:=length(st);

for i:=0 to 255 do y[i]:=0;

for i:=l2 downto 1 do

y[l2-i+1]:=ord(st[i])-ord('0');

if l1<l2 then l1:=l2;

for i:=1 to l1 do

begin

x[i]:=x[i]+y[i];
```

```
x[i+1]:=x[i+1]+x[i] div 10;

x[i]:=x[i] mod 10;

end;

write('x+y=');

j:=255;

while x[j]=0 do j:=j-1;

for i:=j downto 1 do write(x[i]);

readln;

end.
```

四、优化：

以上的方法的有明显的缺点：

（1）浪费空间：一个整型变量（-32768~32767）只存放一位（0~9）；

（2）浪费时间：一次加减只处理一位；

针对以上问题，我们做如下优化：一个数组元素存放四位数；（integer的最大范围是32767，5位的话可能导致出界）将标准数组改为紧缩数组。第一步的具体方法：

```
l:=length(s1);

k1:=260;

repeat {————有关字符串的知识}

s:=copy(s1,l-3,4);

val(s,a[k1],code);

k1:=k1-1;

s1:=copy(s1,1,l-4);

l:=l-4;

until l<=0;

k1:=k1+1;
```

而因为这个改进，算法要相应改变：

（1）运算时：不再逢十进位，而是逢万进位（mod 10000; div 10000);

（2）输出时：最高位直接输出，其余各位，要判断是否足够4位，不足部分要补0；例如：1，23，2345这样三段的数，输出时，应该是100232345而不是1232345。

改进后的算法：

```
var a,b:string; k,i,c,d:longint; e,z,y:array[0..255]of integer;
begin
readln(a);
readln(b);
if length(b)>length(a) then for i:=1 to length(b)-length(a) do
a:='0'+a
else for i:=1 to length(a)-length(b) do
b:='0'+b;
for i:=length(a) downto 1 do
begin
c:=ord(a[i])-48;
d:=ord(b[i])-48;
if c+d<10 then e[i]:=e[i]+c+d else begin e[i]:=e[i]+c+d-10;e[i-1]:=1; end;
end;
if e[0]=1 then k:=0 else k:=1;
for i:=k to length(a) do
write(e[i]);
```

end.

C++参考程序：

```
#include<iostream>
#include<cstdio>
#include<cstring>
using namespace std;
int main()
{
char a1[100],b1[100];
int a[100],b[100],c[100],lena,lenb,lenc,i,x;
memset(a,0,sizeof(a)); memset(b,0,sizeof(b)); memset(c,0,sizeof(c)); gets(a1); gets(b1); //输入加数与被加数
lena=strlen(a1); lenb=strlen(b1); for (i=0;i<=lena-1;i++) a[lena-i]=a1[i]-48; //加数放入a数组 for (i=0;i<=lenb-1;i++) b[lenb-i]=b1[i]-48; //加数放入b数组 lenc =1; x=0; while (lenc <=lena||lenc <=lenb) { c[lenc]=a[lenc]+b[lenc]+x; //两数相加
x=c[lenc]/10; c[lenc]%=10; lenc++; } c[lenc]=x; if (c[lenc]==0) lenc--; //处理最高进位 for (i=lenc;i>=1;i--) cout<<c[i]; //输出结果 cout<<endl;
return 0; }
```

高精度减法

和高精度加法相比，减法在差为负数时处理的细节更多一点：当被减数小于减数时，差为负数，差的绝对值是减数减去被减数；在程序实现上用一个变量来存储符号位，用另一个数组存差的绝对值。

算法流程:(1).读入被减数S1，S2（字符串）；

(2).置符号位：判断被减数是否大于减数：大则将符号位置为空；小则将符号位置为“- ”，交换减数与被减数；

(3).被减数与减数处理成数值，放在数组中；

(4).运算：A、取数；

B、判断是否需要借位；

C、减，将运算结果放到差数组相应位中；

D、判断是否运算完成：是，转5；不是，转A；

(5).打印结果：符号位，第1位，循环处理第2到最后一位；

细节：▲如何判断被减数与减数的大小？

如果位数一样，直接比较字符串大小;否则，位数多的大。

```
k1:=length(s1); k2:=length(s2);
if k1=k2 then
if s1<s2 then begin fh:='-'; s:=s1;s1:=s2; s2:=s;end
else if k1<k2 then begin fh:='-';s:=s1;s1:=s2;s2:=s;end;{s1存被减数，fh存符号}
```

▲将字符串处理成数值：

```
l:=length(s1);{求出s1的长度，也即s1的位数；有关字符串的知识。}
k1:=260;
for i:=l downto 1 do
begin
a[k1]:=ord(s1[i])-48;{将字符转成数值}
k1:=k1-1;
end;
k1:=k1+1;
```

▲运算（减法跟加法比较，减法退位处理跟加法进位处理不一样）：

处理退位： 跟加法一样，在for语句外面先将退位清零，用被减数再减去退位，{注意：由于每一个数位不一定都得向前一位借位，所以这里退位得清零。例如，234-25，个位需借位，而十位不用} 接着，再判断，当被减数某一位不够减时，则需加上前一位退位过来的数。注意：由于这里采用优化方法，所以退一位，就等于后一位加上10000。）最后，再拿一个数组来存储两个减数的差。

```
jw:=0;
for i:=260 downto k1 do
begin
a[i]:=a[i]-jw;{此处jw为从刚处理的那一位上从本一位上的借位}
jw:=0; {此处jw为I 位准备向高一位的借位}
if a[i]<b[i] then
begin
jw:=1;
a[i]:=a[i]+10000;
end;
c[i]:=a[i]-b[i]
end;
```

▲打印结果： 先找到差的第一个非零数，如果差的所有位数都为零，就直接输出零； 如果不是，就输出符号位和差的第一位。剩下部分，打印补足零；因为优化后的高精度减法,是把每四个数位分成一段,而每一段则必须有四个数,当有一段不足四个数时,就得用"0"补足.(如:第一位是'1',第二位是'34',第三位是'345',第四位是'8', 则应写为'').注意:第一位不用补零,(如:第一位为'3',则写成'3').

```
while (c[k]=0) and (k<=260) do k:=k+1;
if k>260 then write('0')
else begin
write(fh,c[k]);{k是差的第1位；}
for i:=k+1 to 260 do
begin
if c[i]<100 then write('0');
if c[i]<10 then write('0');
write(c[i]);
end;
end;
```

参考程序：

```
program ZDloveQC;
var s1,s2,s3,s4,s:string;
a,b,c:array[1..260]of integer;
i,k1,k2,l,code,jw:longint;
fh:string;
begin
readln(s1); readln(s2);
k1:=length(s1); k2:=length(s2); fh:='';
if k1=k2 then
```

```
if s1<s2 then begin fh:='-';s:=s1; s1:=s2; s2:=s; end;
if k1<k2 then begin fh:='-';s:=s1; s1:=s2; s2:=s; end;
k1:=260;
l:=length(s1);
repeat
s3:=copy(s1,l-3,4);
val(s3,a[k1],code);
dec(k1);
s1:=copy(s1,1,l-4);
l:=l-4;
until l<=0;
inc(k1);
l:=length(s2);
k2:=260;
repeat
s4:=copy(s2,l-3,4);
val(s4,b[k2],code);
dec(k2);
s2:=copy(s2,1,l-4);
l:=l-4;
until l<=0;
inc(k2);
jw:=0;
for i:=260 downto k1 do
begin
a[i]:=a[i]-jw;
jw:=0;
if a[i]<b[i] then
begin
jw:=1;
a[i]:=a[i]+10000;
end;
c[i]:=a[i]-b[i];
end;
while (c[k1]=0)and(k1<260) do inc(k1);
if k1>260 then writeln('0')
else begin
write(fh,c[k1]);
for i:=k1+1 to 260 do
begin
```

if c[i]<1000 then write('0');

if c[i]<100 then write('0');

if c[i]<10 then write('0');

write(c[i]);

end;

end;

end.

C++参考程序：

#include<stdio.h>

#include<ctype.h>

#include<string.h>

#include<stdlib.h>

#include<stdbool.h>

int const n=1000;

typedef int arr[n];

int c[2*n+1]={}, i,j,k; arr a,b;void dushu(arr &s){int i=0,j,k,m;
 char b[400],ch;
 scanf("%c",&ch);
 while ((ch>=48)&& (ch <=57))
 { i=i+1;
 b[i]=ch;
 scanf("%c",&ch);
 } k=0; for(j=i; j>=1; j=j-1) { k=k+1; s[k]=b[j]-48; } }void shuchu(int c[n]) { int i=n; j=0; while ((c[i]==0) && (i>1)) i--; for (j=i; j>0; j--) printf("%d",c[j]); printf("\n"); } bool compare(arr &a,arr &b){ int i,j,k; for (i=n; i>0; i--) { if (a[i]>b[i]) {return true;} else if (a[i]<b[i]) {return false;}; } } void change(arr &a, arr &b) { int i,t; for(i=1; i<=n; i++) { t=a[i]; a[i]=b[i]; b[i]=t; } } void jiafa(arr a,arr b) {int i,j,k,m,s,t=0;
 for (i=1; i<=n+1; i++)
 {
 s=a[i]+b[i]+t;
 c[i]=s % 10;
 t=s / 10;
 } shuchu(c); } void jianfa(arr &a, arr &b) { int i,j,k,s; memset(c,0,2*n+1); for (i=1; i<=n; i++) { if (a[i]>=b[i]) c[i]=a[i]-b[i]; else { c[i]=a[i]-b[i]+10; a[i+1]--; } } shuchu(c); }void chengfa(arr a, arr b) { int i,j,k,m; memset(c,0,2*n+1); for (i=1; i<=n; i++) for (j=1; j<=n; j++) { m=i+j-1; c[m]=a[j]*b[i]+c[m]; c[m+1]=c[m] / 10; c[m]=c[m]% 10; } shuchu(c); } main(){ dushu(a); dushu(b); jiafa(a,b); if (!compare(a,b)) { printf("%c",'-'); change(a,b); } jianfa(a,b); chengfa(a,b); system("pause"); return 0;}

单精度乘法

单精度乘法是计算范围次于高精度乘法的一种运算，只是运算效率比高精度计算略高。

单精度乘法过程样例：

const

maxcount=进制位

maxlen=记录高精度数组大小

procedure mulnum(a:bignum;x:longint;,var c:bignum);

var

i:longint;

begin

fillchar(c,sizeof(c),0);c[0]:=a[0];

for i:=1 to c[0] do c[i]:=a[i]*x;

for i:=1 to c[0] do {进位}

begin

inc(c[i+1],c[i] div maxcount);

c[i]:=c[i] mod 10;

end;

while c[c[0]+1]>0 do

begin

```
inc(c[0]);
inc(c[c[0]+1],c[c[0]] div maxcount);
c[c[0]]:=c[c[0]] mod maxcount;
end;
end;
```

高精度乘法

高精度乘法基本思想和加法一样。其基本流程如下：

①读入被乘数s1，乘数s2

②把s1、s2分成4位一段，转成数值存在数组a,b中；记下a,b的长度k1,k2；

③i赋为b中的最低位；

④从b中取出第i位与a相乘，累加到另一数组c中；（注意：累加时错开的位数应是多少位？）

⑤i:=i-1；检测i值：小于k2则转⑥,否则转④

⑥打印结果

参考程序：

```
program chengfa;
const n=100;
type ar=array [1..n] of integer;
var a,b:ar; k1,k2,k:integer;
c:array [1..200] of integer;
s1,s2:string;
procedure fenge(s:string;var d:ar; var kk:integer); {将s分割成四位一组存放在d中，返回的kk值指向d的最高位}
var ss:string;
i,code:integer;
begin
i:=length(s);
kk:=n;
repeat
ss:=copy(s,i-3,4);
val(ss,d[kk],code);
kk:=kk-1;
s:=copy(s,1,i-4);
i:=i-4;
until i<0;
kk:=kk+1;
end;
procedure init;
var i:integer;
begin
for i:=1 to n do begin a:=0; b:=0; end;
for i:=1 to 2*n do c:=0;
```

```
write('input 2 numbers:');
readln(s1);
readln(s2);
fenge(s1,a,k1);
fenge(s2,b,k2);
end;
procedure jisuan;
var i,j,m:integer; x,y,z,jw:longint;
begin
i:=n; k:=2*n;
repeat
x:=b; z:=0; m:=k; jw:=0;
for j:=n downto k1 do
begin
y:=a[j];
z:=c[m];
x:=x*y+z+jw;
jw:=x div 10000;
c[m]:=x mod 10000;
m:=m-1;
x:=b;
end;
if jw<>0 then c[m]:=jw else m:=m+1;
i:=i-1;
k:=k-1;
until i<k2;
k:=m;
end;
procedure daying;
var i:integer;
begin
write(c[k]);
for i:=k+1 to 2*n do
begin
if c<1000 then write('0');
if c<100 then write('0');
if c<10 then write('0');
write(c);
end;
writeln;
```

```
end;
begin
init;
jisuan;
daying;
end.
```

教材“基础编”P87高精乘法参考程序：

```
program ex3_1;
var
a,b,c:array[0..1000] of word;
procedure init;
var
s:string;
ok,i,j:integer;
begin
readln(s);
a[0]:=length(s);
for i:=1 to a[0] do
val(s[a[0]-i+1],a,ok);
readln(s);
b[0]:=length(s);
b[0]:=length(s);
for i:=1 to b[0] do
val(s[b[0]-i+1],b,ok);
end;
procedure highmul;
var i,j,k:integer;
begin
c[0]:=a[0]+b[0];
for i:=1 to b[0] do
for j:=1 to a[0]+1 do
begin
inc(c[i+j-1],a[j]*b mod 10);
c[i+j]:=c[i+j]+(a[j]*b div 10)+(c[i+j-1] div 10);
c[i+j-1]:=c[i+j-1] mod 10;
end;
end;
procedure print;
var i:integer;
begin
```

```
while c[c[0]]=0 do dec(c[0]);

for i:=c[0] downto 1 do

write(c);

writeln;

end;

begin

init;

highmul;

print;

end.
```

C++参考程序：

```
#include<iostream> #include<cstring> #include<cstdio> using namespace std; int main() { char a1[100],b1[100]; int a[100],b[100],c[100],lena,lenb,lenc,i,j,x; memset(a,0,sizeof(a)); memset(b,0,sizeof(b)); memset(c,0,sizeof(c)); gets(a1);gets(b1); lena=strlen(a1);lenb=strlen(b1); for (i=0;i<=lena-1;i++) a[lena-i]=a1[i]-48; for (i=0;i<=lenb-1;i++) b[lenb-i]=b1[i]-48; for (i=1;i<=lena;i++) { x=0; //用于存放进位 for (j=1;j<=lenb;j++) //对乘数的每一位进行处理 { c[i+j-1]=a[i]*b[j]+x+c[i+j-1]; //当前乘积+上次乘积进位+原数 x=c[i+j-1]/10; c[i+j-1] %= 10; } c[i+lenb]=x; //进位 } lenc=lena+lenb; while (c[lenc]==0&&lenc>1) //删除前导0 lenc--; for (i=lenc;i>=1;i--) cout<<c[i]; cout<<endl; return 0; }
```

高精度除法

```
#include<iostream>
#include<fstream>
#include<cmath>
#include<cstdio>
#include<algorithm>
using namespace std;
const long long ans_ary=500;//修改ans_ary的值以改变输出数据精度
long long ans[ans_ary]={0};

void get_shang(int k){//高精度累加
    for(long i=0;i<=ans_ary;i++){
        int jinwei=int(ans[i]/100000);
        ans[i]*=10;
        ans[i]=(ans[i]-jinwei*1000000);
        ans[i+1]+=jinwei;
    }
    ans[0]+=k;
    return;
}
void print(){//输出
    for(int i=0;i<ans_ary;i++){
        int jinwei=0;
        jinwei=int(ans[i]/100000);
        ans[i+1]+=jinwei;
        ans[i]-=100000*jinwei;
    }
    ans[0]/10;
    long long cnt=0;
    for(int i=ans_ary-1;i>=0;i--){
        cout<<"line"<<' '<<cnt+1<<' '<<"count"<<' '<<cnt*5+1<<' ';
        cnt++;
        if(ans[i]==0)cout<<"00000";
        else if(ans[i]<10)cout<<"0000"<<ans[i];
        else if(ans[i]<100)cout<<"000"<<ans[i];
        else if(ans[i]<1000)cout<<"00"<<ans[i];
        else if(ans[i]<10000)cout<<'0'<<ans[i];
        else if(ans[i]>=10000)cout<<ans[i];
        cout<<endl;
    }
}
int main(){//主函数
    freopen("output.out","w",stdout);//会在编译完成的应用程序所在位置生成一个“output.out”的文件
    long long n,m;
    cin>>m>>n;//被除数，除数（m/n）
    long long ready=m;
    for(long long i=0;i<=ans_ary*5;i++){
        long long rest=0;
        long long shang=int(ready/n);
        get_shang(shang);
        ready=ready-shang*n;
        ready=ready*10;
    }
    print();
    system("pause");

    return 0;
}
```

高精度除法：

1）.高精度除以整型数据(integer);

程序如下:

```
program HighPrecision3_Multiply1;
```

```
const
fn_inp='hp5.inp';
fn_out='hp5.out';
maxlen=100; { max length of the number }
type
hp=record
len:integer; { length of the number }
s:array[1..maxlen] of integer
{ s[1] is the lowest position
s[len] is the highest position }
end;
var
x,y:hp;
z,w:integer;
procedure PrintHP(const p:hp);
var i:integer;
begin
for i:=p.len downto 1 do write(p.s[i]);
end;
procedure init;
var
st:string;
i:integer;
begin
assign(input,fn_inp);
reset(input);
readln(st);
x.len:=length(st);
for i:=1 to x.len do { change string to HP }
x.s:=ord(st[x.len+1-i])-ord('0');
readln(z);
close(input);
end;
procedure Divide(a:hp;b:integer;var c:hp;var d:integer);
{ c:=a div b ; d:=a mod b }
var i,len:integer;
begin
fillchar(c,sizeof(c),0);
len:=a.len;
d:=0;
```

for i:=len downto 1 do { from high to low }

begin

d:=d*10+a.s[i];

c.s:=d div b;

d:=d mod b;

end;

while(len>1) and (c.s[len]=0) do dec(len);

c.len:=len;

end;

procedure main;

begin

Divide(x,z,y,w);

end;

procedure out_;

begin

assign(output,fn_out);

rewrite(output);

PrintHP(y);

writeln;

writeln(w);

close(output);

end;

begin

init;

main;

out_;

end.

2）.高精度除以高精度

程序如下:

版本一：

```
programHighPrecision4;{outputalpha/
beta}
const
    fn_inp='hp6.inp';
    fn_out='hp6.out';
    maxlen=100;{maxlengthofthenumber}
type
    hp=record
        len:integer;{lengthofthenumber}
        s:array[1..maxlen]ofinteger
    {s[1]isthelowestposition
    s[len]isthehighestposition}
        end;
var
    x:array[1..2]ofhp;
    y,w:hp;{x:input;y:output}
procedurePrintHP(constp:hp);
vari:integer;
begin
    fori:=p.lendownto1dowrite(p.s[i]);
end;
procedureinit;
var
    st:string;
    j,i:integer;
begin
    //assign(input,fn_inp);
    //reset(input);
    forj:=1to2do
    begin
```

```
        readln(st);
        x[j].len:=length(st);
        fori:=1tox[j].lendo{changestringtoHP}
        x[j].s[i]:=ord(st[x[j].len+1-i])-ord('0');
    end;
    //close(input);
end;
procedureSubtract(a,b:hp;varc:hp);{c:=a-b,supposea>=b}
vari,len:integer;
begin
    fillchar(c,sizeof(c),0);
    ifa.len>b.lenthenlen:=a.len{getthebiggerlengthofa,b}
        elselen:=b.len;
    fori:=1tolendo{subtractfromlowtohigh}
    begin
        inc(c.s[i],a.s[i]-b.s[i]);
        ifc.s[i]<0then
        begin
            inc(c.s[i],10);
            dec(c.s[i+1]);{add1toahigherposition}
        end;
    end;

while(len>1)and(c.s[len]=0)dodec(len);
    c.len:=len;
end;
functionCompare(consta,b:hp):integer;
{
1ifa>b
0ifa=b
-1ifa<b
}
varlen:integer;
begin
    ifa.len>b.lenthenlen:=a.len{getthebiggerlengthofa,b}
        elselen:=b.len;
    while(len>0)and(a.s[len]=b.s[len])dodec(len);
    {findapositionwhichhaveadifferentdigit}
    iflen=0thencompare:=0{nodifference}
        elsecompare:=a.s[len]-b.s[len];
end;
procedureMultiply10(vara:hp);{a:=a*10}
vari:Integer;
begin
    fori:=a.lendownto1do
a.s[i+1]:=a.s[i];
    a.s[1]:=0;
    inc(a.len);
    while(a.len>1)and(a.s[a.len]=0)dodec(a.len);
end;
procedureDivide(a,b:hp;varc,d:hp);{c:=adivb;d:=amodb}
vari,j,len:integer;
begin
    fillchar(c,sizeof(c),0);
    len:=a.len;
    fillchar(d,sizeof(d),0);
    d.len:=1;
    fori:=lendownto1do
    begin
        Multiply10(d);
        d.s[1]:=a.s[i];{d:=d*10+a.s}
    {c.s:=ddivb;d:=dmodb;}
    {
while(d>=b)dobegind:=d-b;inc(c.s)end}
        while(compare(d,b)>=0)do
        begin
            Subtract(d,b,d);
            inc(c.s[i]);
        end;
    end;

while(len>1)and(c.s[len]=0)dodec(len);
    c.len:=len;
end;
proceduremain;
begin
    Divide(x[1],x[2],y,w);
end;
procedureout;
begin
    //assign(output,fn_out);
    //rewrite(output);
    PrintHP(y);{outputalphadivbeta}

writeln;
    PrintHP(w);{outputalphamodbeta}
    writeln;
    //close(output);
end;
begin
    init;
    main;
    out;
end.
```

版本二：

```
programaaa;
type
big=array[0..500]ofinteger;
var
a,b,c,d:big;
proceduremake;//读入数据
var
s:ansistring;
i:longint;
begin
readln(s);
a[0]:=pos('',s)-1;
fori:=1toa[0]do
a[i]:=ord(s[a[0]-i+1])-ord('0');
delete(s,1,a[0]+1);
b[0]:=length(s);
fori:=1tob[0]do
b[i]:=ord(s[b[0]-i+1])-ord('0');
end;
proceduremulti10(varc:big);//余数乘十
var
i:longint;
```

```
begin
inc(c[0]);
fori:=c[0]downto2do
c[i]:=c[i-1];
ifc[c[0]]=0thendec(c[0]);
end;
functioncompare(vara,b:big):integer;//比较
var
i:longint;
begin
ifa[0]>b[0]thenexit(1);
ifa[0]<b[0]thenexit(-1);
fori:=a[0]downto1do
ifa[i]>b[i]thenexit(1)
else
ifa[i]<b[i]thenexit(-1);
exit(0);
end;
procedureminus(vara,b:big);//减法
var
i:longint;
begin
fori:=1toa[0]do
begin
a[i]:=a[i]-b[i];
ifa[i]<0then
begin
a[i]:=a[i]+10;
a[i+1]:=a[i+1]-1;
end;
end;
while(a[a[0]]=0)and(a[0]>1)do
dec(a[0]);
end;
proceduredivision(vara,b,c,d:big);//
除法,c为商，d为
余数。
var
i:longint;
begin
fillchar(c,sizeof(c),0);

fillchar(d,sizeof(d),0);
d[0]:=1;
fori:=a[0]downto1do
begin
multi10(d);
d[1]:=a[i];
whilecompare(d,b)>-1do
begin
minus(d,b);
inc(c[i]);
end;
end;
c[0]:=a[0];
while(c[c[0]]=0)and(c[0]>1)do
dec(c[0]);
end;
procedureprint(vara:big);
var
i:longint;
begin
fori:=a[0]downto1do
write(a[i]);

writeln;
end;
begin
make;
division(a,b,c,d);
print(c);
{print(d);}
readln;
readln;
end.
```

高精度阶乘

作为一种高精度乘法的扩展算法，实质为高精度乘低精度，算法如下： var

a:array[1..10000] of longint;

i,j,k,l,p,o,q,x,y,w:integer;

begin

read(i);

a[1]:=1;

w:=1;

for j:=1 to i do

begin

y:=0; //到“For”前可省，但改为for k:=1 to 10000 do

x:=j;

while x>0 do

begin

y:=y+1;

```
x:=x div 10;

end;

o:=0;

for k:=w to l+y+1 do

begin

q:=a[k]*j+o;

o:=q div 10;

a[k]:=q mod 10;

end;

l:=10000;

while (a[l]=0) and (l>1) do l:=l-1;

w:=1;

while (a[w]=0) and (w<9999) do w:=w+1;

end;

for p:=l downto 1 do

write(a[p]);

writeln;

end.
```

C++的优雅实现

编辑

我们知道，C++是一个面向对象的语言。上述所有代码的实现都是面向过程的，都是以高精度运算为主体进行编程。然而，在实际应用中，高精度通常只作为程序的一部分而出现，在这样的情况下，上述代码难以直接移植、使用的特性暴露无遗。我们用C++的面向对象编程特性来做一次非常好用的高精度。

实现

我们使用标准库vector做基类，完美解决位数问题，同时更易于实现。 [1]

高精度类型Wint包含一个低精度转高精度的初始化函数，可以自动被编译器调用，因此无需单独写高精度数和低精度数的运算函数，十分方便；还包含了一个在各类运算中经常用到的进位小函数。

```
#include<iostream>
#include<vector>
#include<string>
using namespace std;
struct Wint:vector<int>
{
    Wint(int n=0)//默认初始化为0，但0的保存形式为空
    {
        push_back(n);
        check();
    }
    Wint& check()
    {
        while(!empty()&&!back())pop_back();//去除最高位可能存在的0
        if(empty())return *this;
        for(int i=1; i<size(); ++i)
        {
            (*this)[i]+=(*this)[i-1]/10;
            (*this)[i-1]%=10;
        }
        while(back()>=10)
```



```
    }
};
```

输入输出

平淡无奇，有很多读入的方法。这里偷个懒，直接读入一个字符串再转入Wint中。

```
istream& operator>>(istream &is,Wint &n)
{
    string s;
    is>>s;
    n.clear();
    for(int i=s.size()-1; i>=0; --i)n.push_back(s[i]-'0');
    return is;
}
ostream& operator<<(ostream &os,const Wint &n)
```

```
{
    if(n.empty())os<<0;
    for(int i=n.size()-1; i>=0; --i)os<<n[i];
    return os;
}
```

大小比较

比较，只需要写两个，其他的直接代入即可。值得注意的是，这里用常量引用当参数，避免拷贝更高效。

```
bool operator!=(const Wint &a,const Wint &b)
{
    if(a.size()!=b.size())return 1;
    for(int i=a.size()-1; i>=0; --i)
        if(a[i]!=b[i])return 1;
    return 0;
}
bool operator==(const Wint &a,const Wint &b)
{
    return !(a!=b);
}
bool operator<(const Wint &a,const Wint &b)
{
    if(a.size()!=b.size())return a.size()<b.size();
    for(int i=a.size()-1; i>=0; --i)
        if(a[i]!=b[i])return a[i]<b[i];
    return 0;
}
bool operator>(const Wint &a,const Wint &b)
{
    return b<a;
}
bool operator<=(const Wint &a,const Wint &b)
{
    return !(a>b);
}
bool operator>=(const Wint &a,const Wint &b)
{
    return !(a<b);
}
```

加法

加法，先实现+=，这样更简洁高效。注意各个参数有别。

```
Wint& operator+=(Wint &a,const Wint &b)
{
    if(a.size()<b.size())a.resize(b.size());
    for(int i=0; i!=b.size(); ++i)a[i]+=b[i];
    return a.check();
}
Wint operator+(Wint a,const Wint &b)
{
    return a+=b;
}
```

减法

减法，返回差的绝对值，由于后面有交换，故参数不用引用。

```
Wint& operator-=(Wint &a,Wint b)
{
    if(a<b)swap(a,b);
    for(int i=0; i!=b.size(); a[i]-=b[i],++i)
        if(a[i]<b[i])//需要借位
        {
            int j=i+1;
            while(!a[j])++j;
            while(j>i)
            {
                --a[j];
                a[--j]+=10;
            }
        }
    return a.check();
}
Wint operator-(Wint a,const Wint &b)
{
    return a-=b;
}
```

乘法

乘法不能先实现*=，原因自己想。

```
Wint operator*(const Wint &a,const Wint &b)
{
    Wint n;
    n.assign(a.size()+b.size()-1,0);
    for(int i=0; i!=a.size(); ++i)
        for(int j=0; j!=b.size(); ++j)
            n[i+j]+=a[i]*b[j];
    return n.check();
}
Wint& operator*=(Wint &a,const Wint &b)
{
    return a=a*b;
}
```

除法和取模

除法和取模先实现一个带余除法函数。当然，高精度除法也可以用二分档案法实现，不过效率过低且代码冗长，这里使用常规竖式除法。

```
Wint divmod(Wint &a,const Wint &b)
{
    Wint ans;
    for(int t=a.size()-b.size(); a>=b; --t)
    {
```

```
        Wint d;
        d.assign(t+1,0);
        d.back()=1;
        Wint c=b*d;
        while(a>=c)
        {
            a-=c;
            ans+=d;
        }
    }
    return ans;
}
Wint operator/(Wint a,const Wint &b)
{
    return divmod(a,b);
}
Wint& operator/=(Wint &a,const Wint &b)
{
    return a=a/b;
}
Wint& operator%=(Wint &a,const Wint &b)
{
    divmod(a,b);
    return a;
}
Wint operator%(Wint a,const Wint &b)
{
    return a%=b;
}
```

使用

通过重载运算符，还可以实现++、--、^、!、逻辑运算符等很多运算，十分简单，此处都不写了。

此时你几乎可以像int一般便捷地使用Wint，甚至可以把Wint和int混合使用。

顺手实现一个快速幂，可以看到和普通快速幂几乎无异。

```
Wint pow(const Wint &n,const Wint &k)
{
    if(k.empty())return 1;
    if(k==2)return n*n;
    if(k.back()%2)return n*pow(n,k-1);
    return pow(pow(n,k/2),2);
}
int main()
{
    Wint a,b;
    //可以把a或b改成int型，仍能正常使用
    cin>>a>>b;
    cout<<(a<b)<<endl
        <<(a==b)<<endl
        <<a+b<<endl
        <<a-b<<endl
        <<a*b<<endl
        <<a/b<<endl
        <<a%b<<endl
        <<pow(a,b);
}
```

优化与改进

上述高精度代码已经能满足正常使用需求了，不过仍然有优化和改进的空间：

一、万进制优化

用int保存个位数显然太过浪费，short型运算效率又没有int型高（绝大部分机器对int有特别优化），在这样的情况下，我们将改进后的Wint每位保存十进制下的四位（首位可能有前导0）即万进制。在这样的优化下，空间占用四分之一，加法快4倍，乘法16倍，而除法可达64倍之多。当然，这仍属于常数级优化，不过底层运算十分频繁的情况下还是值得考虑的。上述代码无需作出太大调整，只需输入输出、进位、减法除法等函数略加改进即可，代码略。

二、低精度优化

前面说过，目前高精度和低精度的运算会先将低精度提升到高精度再进行运算，这就有了优化空间。注意，这里的优化低精度是基于Wint是由int组成这一特点进行的，大于int型的别的类型（如long long）仍需提升到Wint再运算。考虑到低精度数位数在十位以内，优化后效率提升其实不到十倍。不过，在高精度和低精度混合运算十分频繁的情况下，专门写优化的高精度和低精度的运算也聊胜于无。这里先给出加法的优化示例。

```
Wint& operator+=(Wint &a,const int &b)
{
    if(a.empty())a.push_back(b);
    else a.front()+=b;
    return a.check();
}
Wint operator+(Wint a,const int &b)
{
    return a+=b;
}
Wint operator+(const int &b,Wint a)//注意重载不同顺序的运算符
{
    return a+=b;
}
```

三、初始化函数改进：

注意到初始化函数Wint(int n)会将大于int表示范围的类型数如（unsigned long long）先转为低精度再初始化，我们再增加（或者直接替换原先的）初始化函数，改为：

```
Wint(unsigned long long n=0)//需写在Wint定义内
{
    while(n)
    {
```

```
        push_back(n%10);
        n/=10;
    }
}
```

此外，在代码中如果想定义到一个高精度常量（20位向上），就必须增加一个字符串初始化函数而不能直接赋一个整型常数（想想为什么？）。但是，这个字符串初始化函数我们希望它不会像int型一样在运算中自动提升至Wint。否则，像s+7这样的表达式就会有意义（先将s隐形转换至Wint再进行加法运算，返回一个Wint型结果），但通常不符合我们的预期而仅仅只是代码错误，而编译时无法找出，会给我们调试代码带来很大麻烦。所以，这个字符串初始化函数前需加关键字explicit，来指出我们不希望隐式转换的发生。

```
explicit Wint(const string &s)//需写在Wint定义内
{
    for(int i=s.size()-1;i>=0;--i)push_back(s[i]-'0');//未判断字符串合法情况
}//现在你可以直接用字符串初始化Wint了，改进后输入函数还能这样写
istream& operator>>(istream &is,Wint &n)
{
    string s;
    is>>s;
    n=Wint(s);
    return is;
}
```

参考资料

1. 高精度算法 . 百度百科[引用日期2016-10-30]