# 高精度

## 高精度比较：

### 1）思路：

高精度比较大小的步骤大致如下：

1、比较两个数的长度，长度更长的数越大。

2、如果两个数长度相等，那么就从高位到低位一位一位比较，如果某一位数字不同时，较大的数大。否则继续比较下一位。

3、如果比到最后都没有比出谁大谁小，就说明这两个数一样大。

### 2）代码：

int a[6666],b[6666];

int compare(){

if(lena>lenb) return 1;//lena表示a这个数的长度，lenb则表示b的长度

if(lenb>lena) return 0;//步骤1

for(int i=lena;i>0;i--){//从高位到底位一位一位比较

if(a[i]>b[i]) return 1;

if(b[i]>a[i]) return 0;

}//步骤2

return 0;//步骤3，a=b，即a不大于b

}

## 高精度加法：

### 1）思路:

这一题属于高精度加法。基本思路是先储存字符串，把字符串转化为数组。然后从个位开始相加。每一位相加时，需要处理进位问题。每次相加后进位部分加到下一位，除以10的余数部分保留下来。最后输出答案。

### 2)代码：

#include<iostream>

#include<cstring>

#include<string>

#include<cmath>

#include<algorithm>

using namespace std;

const int N = 20000;

int a[N], b[N], c[N];

int lena, lenb, lenc;

char s1[N], s2[N];

int main() {

cin >> s1 >> s2;

lena = strlen(s1);

lenb = strlen(s2);

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

a[i] = s1[lena - i - 1] - '0';

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

b[i] = s2[lenb - i - 1] - '0';

lenc = max(lena, lenb);

for (int i = 0; i < lenc; i++) {

c[i] = c[i] + a[i] + b[i];

c[i + 1] = c[i] / 10;

c[i] = c[i] % 10;

}

while (c[lenc] > 0)

lenc++;

for (int i = lenc - 1; i >= 0; i--)

cout << c[i];

}

## 高精度减法：

### 思路：

减法与加法相似，但也有所不同，减法不用进位，但要考虑借位。同时还要考虑符号的正负。因此我们可以先判断符号的正负，然后再相减。相减时，可以令较大的数为被减数，较小的数为减数，然后相减。最后输出。

### 代码：

#include<iostream>

#include<cstring>

#include<string>

#include<cmath>

#include<algorithm>

using namespace std;

const int N = 20000;

int a[N], b[N], c[N];

int lena, lenb, lenc;

char s1[N], s2[N];

int main() {

cin >> s1 >> s2;

lena = strlen(s1);

lenb = strlen(s2);

if (lenb > lena || (lenb == lena && (s2[lenb - 1] > s1[lena - 1]))) {

cout << "-";

swap(s1, s2);

swap(lena, lenb);

}

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

a[i] = s1[lena - i - 1] - '0';

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

b[i] = s2[lenb - i - 1] - '0';

lenc = max(lena, lenb);

for (int i = 0; i < lenc; i++) {

c[i] = c[i] + a[i] - b[i];

if (c[i] < 0) {

c[i] = c[i] + 10;

c[i + 1] = -1;

}

}

while (c[lenc - 1] == 0 && lenc > 1)

lenc--;

for (int i = lenc - 1; i >= 0; i--)

cout << c[i];

}

## 高精度乘法：

### 思路：

乘法与加法相似，又不大一样，加法与乘法都要处理进位问题。但乘法还要处理位数问题。即把一个数与另一个数的每一位数相乘（还要注意位数，即每次相乘后面有几个0）.最后还要把答案前面的多余的零去掉。因为零乘以其他数都为零，所以要另外处理，不然输出的零不止一个（这个错误我找了好久）。

### 代码：

#include<iostream>

#include<cstring>

#include<string>

#include<cmath>

#include<algorithm>

using namespace std;

const int N = 20000;

int a[N], b[N], c[N];

int lena, lenb, lenc;

char s1[N], s2[N];

int main() {

cin >> s1 >> s2;

if (!(strcmp(s1,"0"))||!(strcmp(s1, "0")))

{

cout << "0";

}

else

{

lena = strlen(s1);

lenb = strlen(s2);

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

a[i] = s1[lena - i - 1] - '0';

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

b[i] = s2[lenb - i - 1] - '0';

lenc = lena + lenb - 1;

for (int i = 0; i < lena; i++) {

for (int j = 0; j < lenb; j++) {

c[i + j] += a[i] \* b[j];

c[i + j + 1] += c[i + j] / 10;

c[i + j] %= 10;

}

}

while (c[lenc] > 0)

lenc++;

for (int i = lenc - 1; i >= 0; i--)

cout << c[i];

}

}

## 高精度除法（高除单）：

### 1）思路：

这个是高精度除以单精度的类型，即除数不算特别大，可以被已有的数字类型储存。按照平时算除法的方式。先算最高位，然后得到其余数，接着放到下一位继续计算。

### 代码：

#include<iostream>

#include<cstring>

#include<string>

#include<cmath>

#include<algorithm>

using namespace std;

const int N = 20000;

int a[N], b, r;

int lena;

char s1[N], s2[N];

int main() {

cin >> s1 >> b;

lena = strlen(s1);

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

a[i] = s1[lena - i - 1] - '0';

for (int i = lena - 1; i >= 0; i--) {

r = r \* 10 + a[i];

a[i] = r / b;

r = r % b;

}

while (a[lena - 1] == 0 && lena > 1)

lena--;

for (int i = lena - 1; i >= 0; i--)

cout << a[i];

}

## 高精度求余：

### 思路：

这一道题属于高精度求模的题。求法与高精度除法有点相似，求法主要是从最高位一步步往下求余。

### 代码：

#include<iostream>

#include<cstring>

#include<string>

#include<cmath>

#include<algorithm>

using namespace std;

const int N = 20000;

char m[N];

int a[N];

int main()

{

int n;

cin >> n;

cin >> m;

int lenm = strlen(m);

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

a[i] = m[i] - '0';

long long ans = 0;

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

ans = (ans \* 10 + a[i]) % n;

if (ans == 0)

cout << n << endl;

else

cout << ans << endl;

return 0;

}

## 7.高精度除法（高除高）：