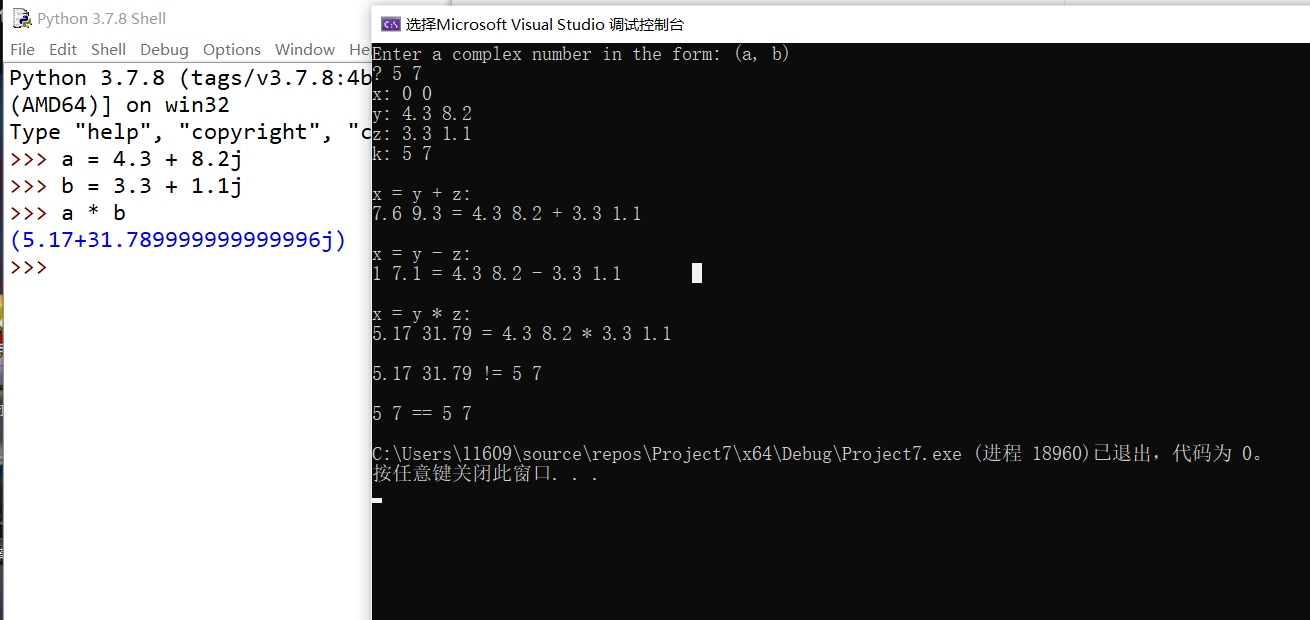
**EX1: Complex Class**



#include<iostream>

#include<cstdio>

using namespace std;

class Complex

{

friend istream& operator>>(istream& cin,Complex& target);

friend ostream& operator<<(ostream& cout, const Complex& target);

double real\_n;

double imag\_n;

public:

Complex(double r = 0, double i = 0):real\_n(r), imag\_n(i){}

Complex operator\*(const Complex& data)const

{

Complex temp; //在RationalNumber那题中尝试直接构造以减少内存读取次数,不过阅读并不友好

temp.real\_n = this->real\_n \* data.real\_n - this->imag\_n \* data.imag\_n;

temp.imag\_n = this->real\_n \* data.imag\_n + this->imag\_n \* data.real\_n;

return temp;

}

Complex operator+(const Complex& data)const

{

Complex temp;

temp.real\_n = this->real\_n + data.real\_n;

temp.imag\_n = this->imag\_n + data.imag\_n;

return temp;

}

Complex operator-(const Complex& data)const

{

Complex temp;

temp.real\_n = this->real\_n - data.real\_n;

temp.imag\_n = this->imag\_n - data.imag\_n;

return temp;

}

bool operator==(const Complex& data) const

{

return ((this->real\_n == data.real\_n) && (this->imag\_n == data.imag\_n));

}

bool operator!=(const Complex& data)const

{

return (!(\*this == data));

}

};

istream& operator>>(istream& cin, Complex& target)

{

cin >> target.real\_n >> target.imag\_n; //原则上应该进行安全性检查并且不应直接修改数据，这里并不知以何为检查标准，所以没有做

return cin;

}

ostream& operator<<(ostream& cout, const Complex& target)

{

cout << target.real\_n << " " << target.imag\_n;

return cout;

}

int main()

{

Complex x, y(4.3, 8.2), z(3.3, 1.1), k;

cout << "Enter a complex number in the form: (a, b)\n? ";

cin >> k; // demonstrating overloaded >>

cout << "x: " << x << "\ny: " << y << "\nz: " << z << "\nk: "

<< k << '\n'; // demonstrating overloaded <<

x = y + z; // demonstrating overloaded + and =

cout << "\nx = y + z:\n" << x << " = " << y << " + " << z << '\n';

x = y - z; // demonstrating overloaded - and =

cout << "\nx = y - z:\n" << x << " = " << y << " - " << z << '\n';

x = y \* z; // demonstrating overloaded \* and =

cout << "\nx = y \* z:\n" << x << " = " << y << " \* " << z << "\n\n";

if (x != k) // demonstrating overloaded !=

cout << x << " != " << k << '\n';

cout << '\n';

x = k;

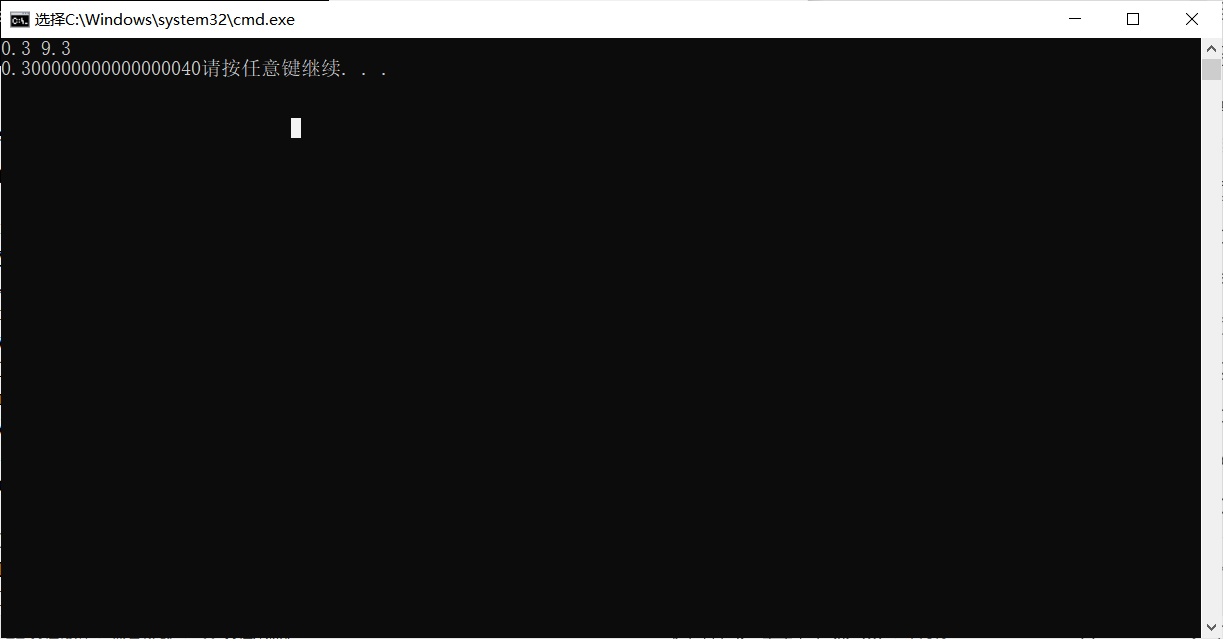
if (x == k) // demonstrating overloaded ==

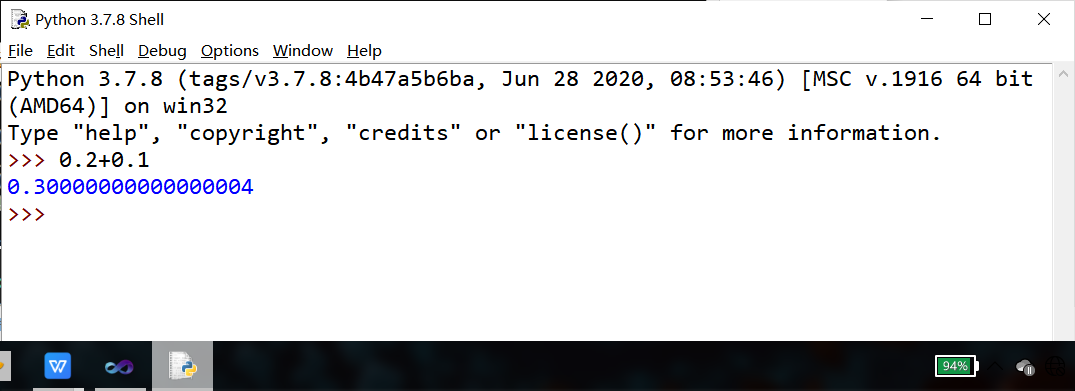
cout << x << " == " << k << '\n';

return 0;

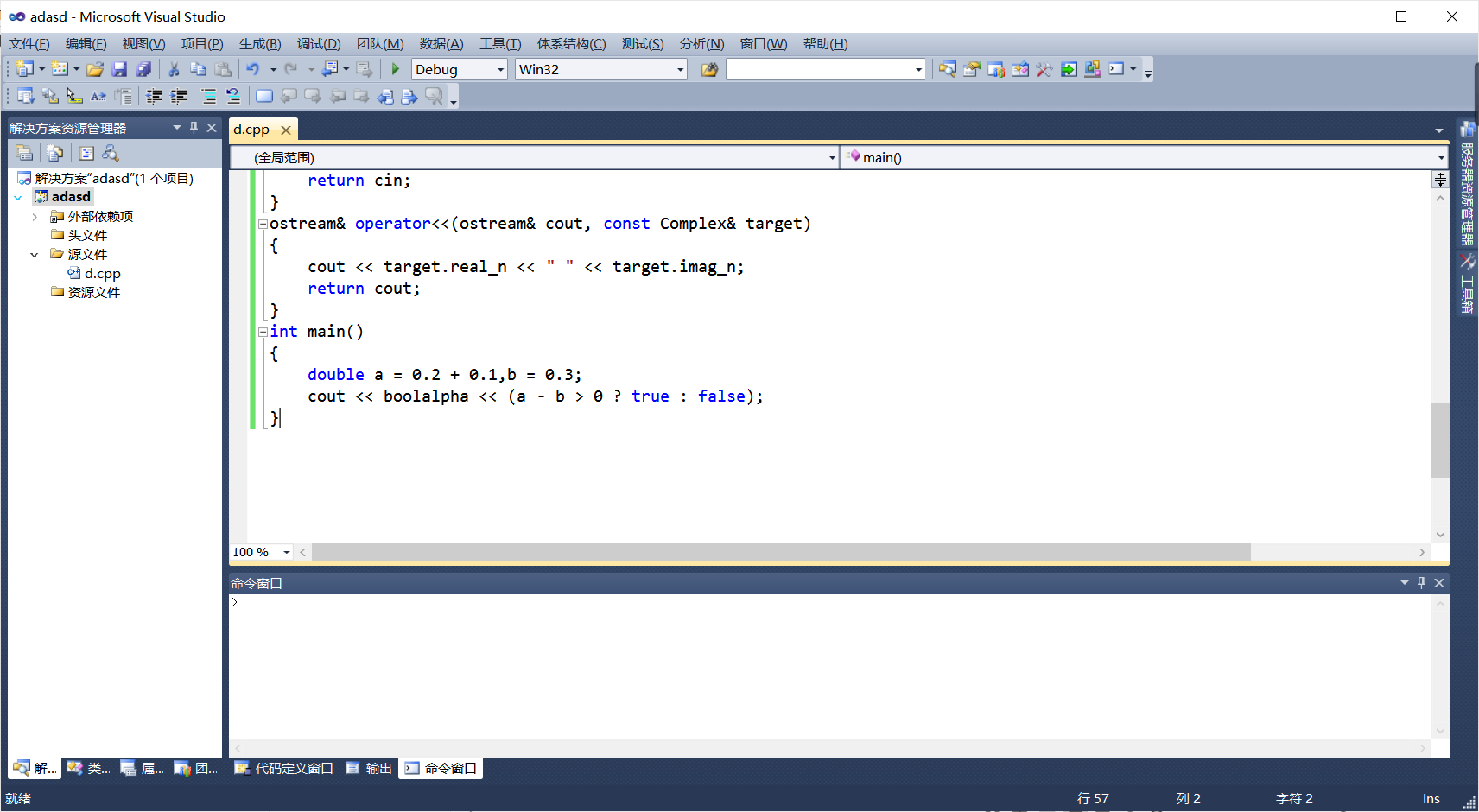
}

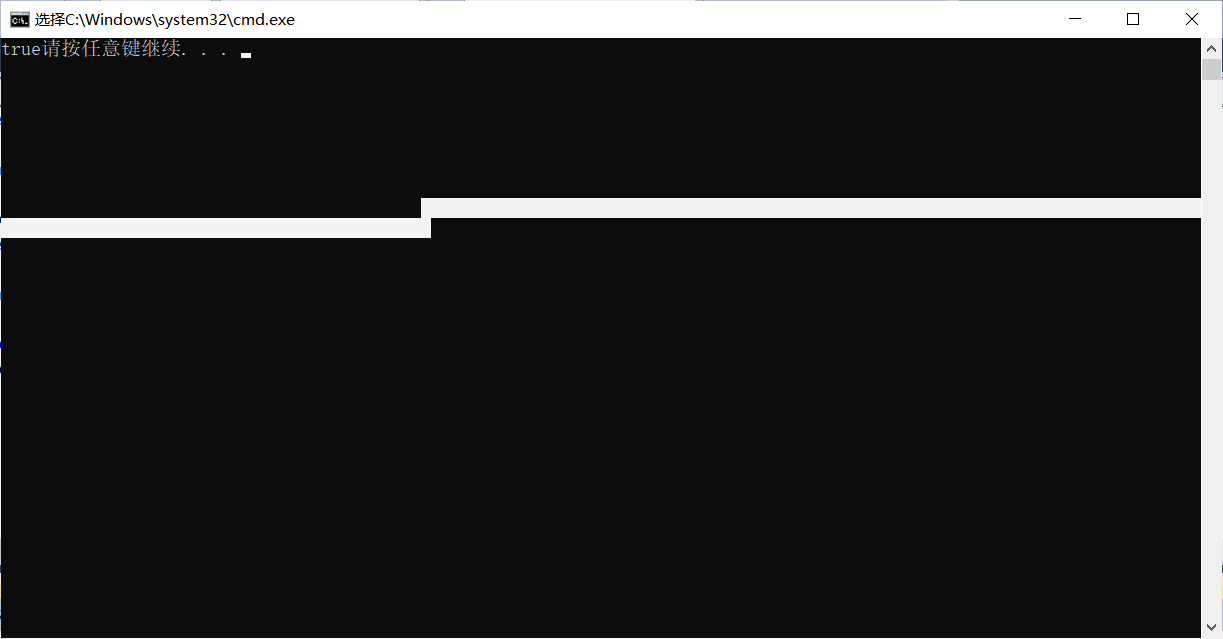
**问题与思考：**





因为浮点数的编码方式（IEEE754），上述问题在追求小数点后高精度时可能出现意料外的问题，这一点需要注意。如下代码体现了这一点。





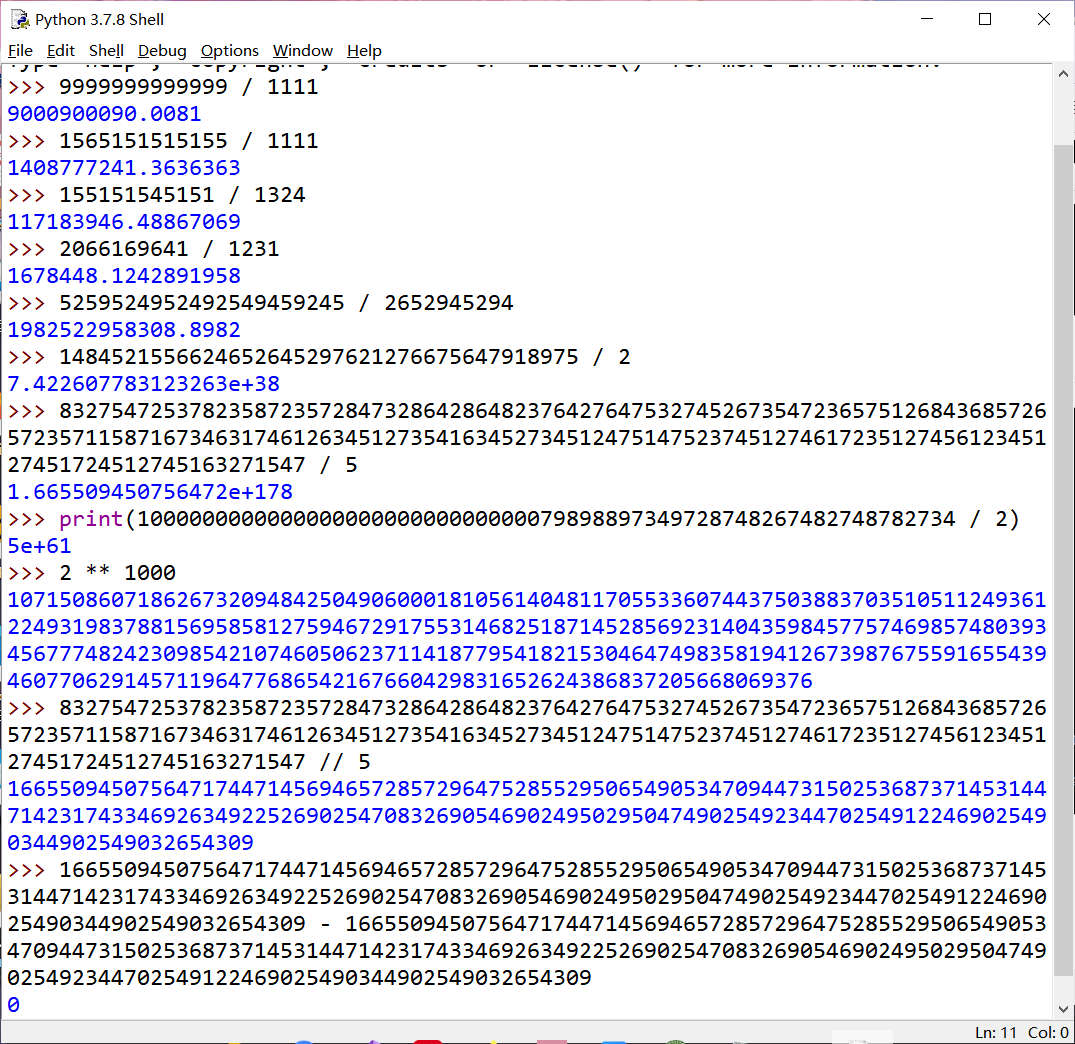
**EX2: Huge Integer Class**

可与下方对照（仅计算除的整数部分）

尝试动态内存实现，出现了一些问题，这里就直接用上次extra提交的实现补充除法。

（采用高精除高精函数）除法：



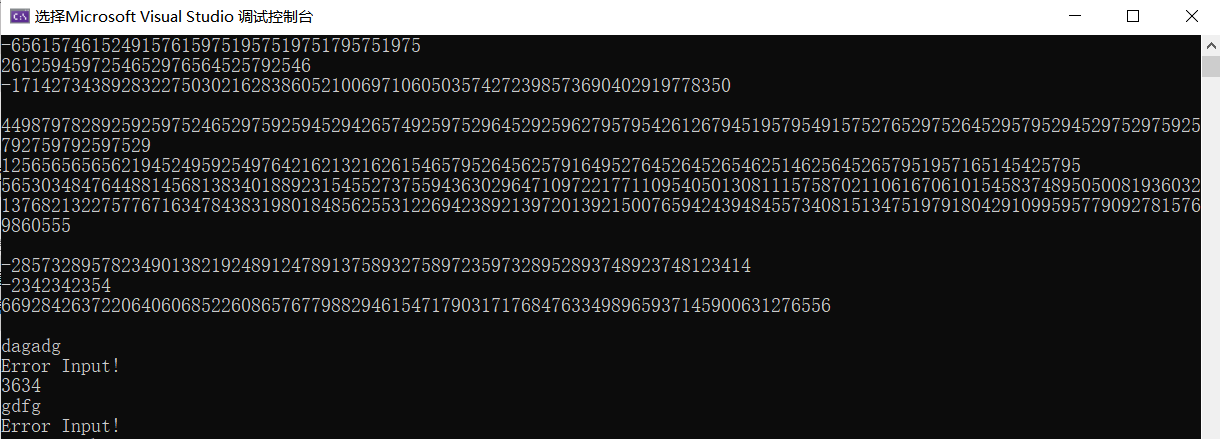


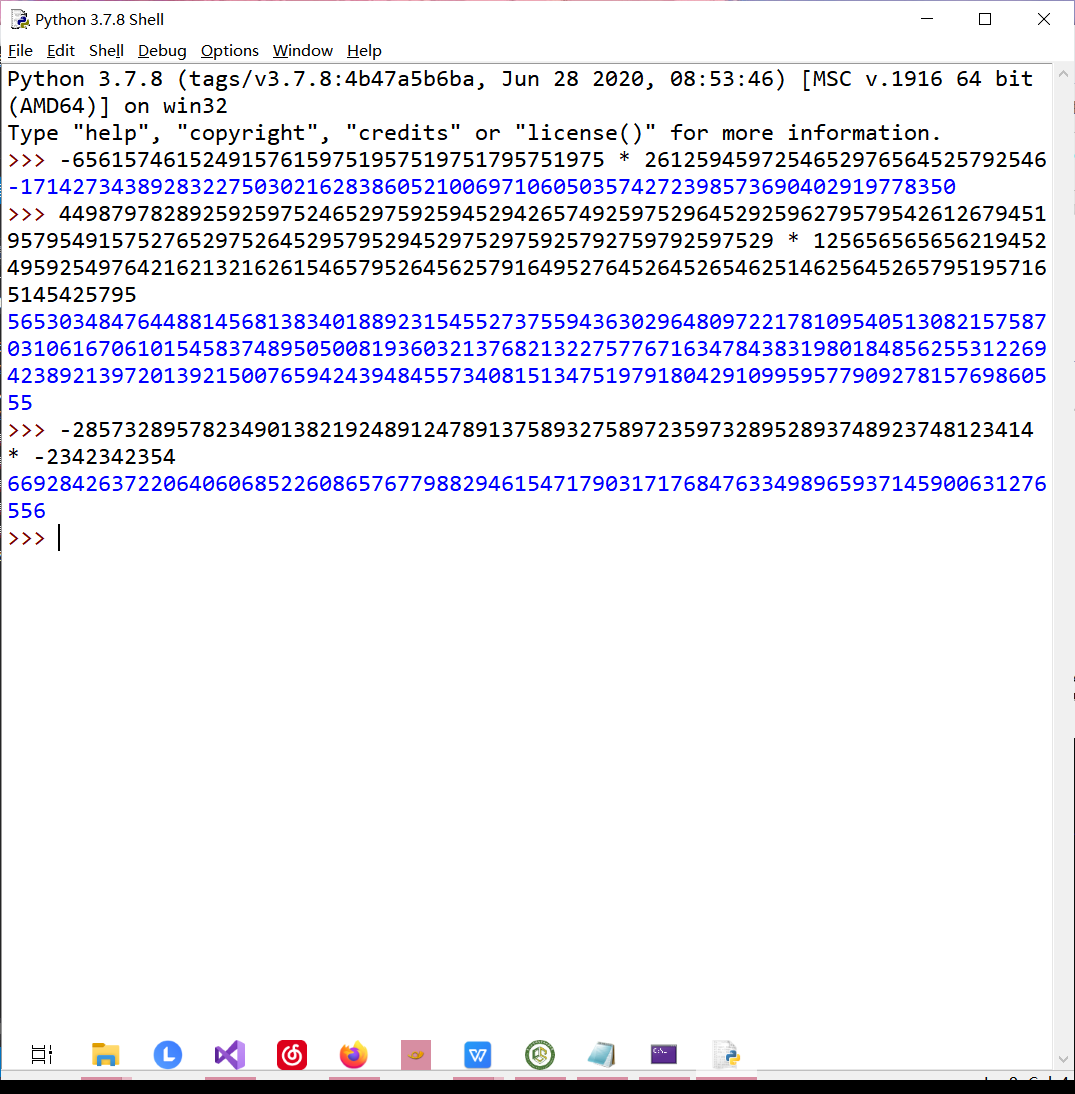
写了三个辅助函数放在类外，放进去也行。

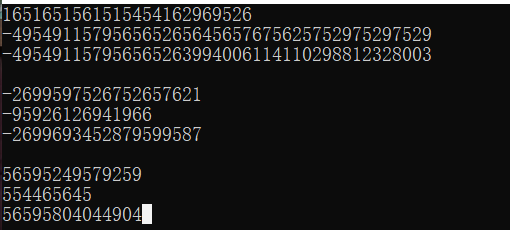
除法没加正负号处理，加减乘都支持负号处理

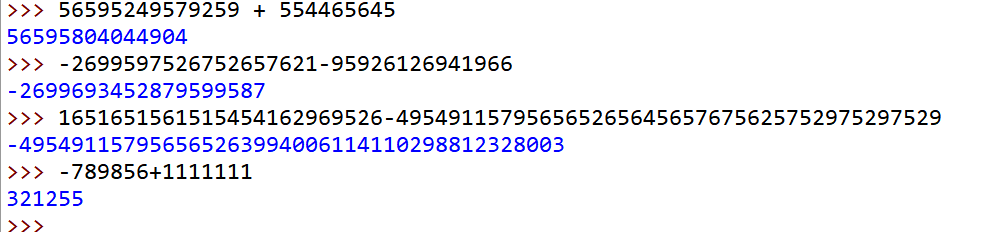
乘法：

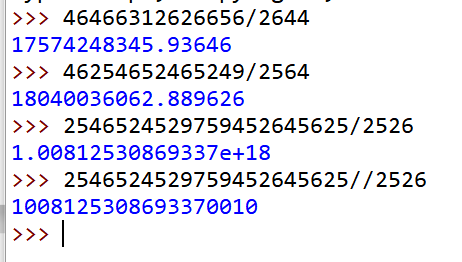
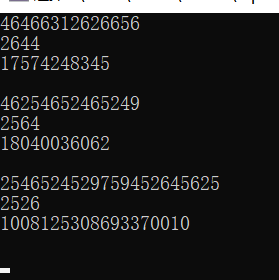
（仅输入与构造警告非法，并未在四则运算中提供合法性处理）





加法（减法）：



（高精除低精函数）：

（下方为VS2019 C++11 VS2010在更下方）

#include<iostream>

#include<vector>

#include<sstream>

#include<string>

#include<regex>

using namespace std;

constexpr const int base = 10000;

bool isBorrow(int i, string& s1, string& s2)

{

return s1[i] - s2[i] < 0;

}

int validize(string& s)

{

int cnt = 0;

for (int i = 0; i < s.size(); i++)

{

if (s[i] == '0') cnt++;

if (s[i] != '0') break;

}

if (cnt == s.size()) return 0;

if (!cnt) return 1;

else

{

string temp;

for (int i = cnt; i < s.size(); i++)

temp += s[i];

s = temp;

}

return cnt;

}

string minuss(string& s1, string& s2)

{

int len = s1.size();

string res;

string temp1 = s1;

string temp2 = s2;

if (s1 < s2)

{

res += "-1";

return res;

}

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

{

again:

if (isBorrow(i, temp1, temp2))

{

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

{

if (temp1[j] > '0')

{

temp1[j] -= 1;

temp1[j + 1] += 10;

break;

}

}

if (isBorrow(i, temp1, temp2))

goto again;

}

temp1[i] -= temp2[i];

temp1[i] += 48;

}

res = temp1;

return res;

}

class Int

{

friend ostream& operator<<(ostream& cout, const Int& Num);

friend Int operator-(const Int& Num);

friend istream& operator>>(istream& cin, Int& Num);

vector<int> Num;

bool isminus = 0;

bool isskip = 0;

public:

Int() {};

Int(string &s)

{

int len = s.size();

stringstream ss;

int temp = 0;

if (s[0] != '-' && (s[0] < '0' || s[0] > '9'))

{

Error2:

printf("Invalid constructor\n");

return;

}

else

{

regex pattern("^([1-9]|-)([0-9]\*)");

cmatch res;

if (regex\_match(s.c\_str(), res, pattern));

else goto Error2;

}

if (s[0] == '-')

isminus = 1;

while (len / 4)

{

ss << s.substr(len - 3 - 1, 4);

ss >> temp;

ss.clear();

Num.emplace\_back(temp);

len -= 4;

}

if (len)

{

int i = 0;

if (isminus) isskip = 1;

if (isskip) i = 1, len -= 1;

ss << s.substr(i, len);

ss >> temp;

ss.clear();

Num.emplace\_back(temp);

}

else if (isminus && !len)

Num[Num.size() - 1] = -Num[Num.size() - 1];

}

Int(const char\* c)

{

string s = c;

Int temp(s);

\*this = temp;

}

bool operator>(const Int& Num2) const

{

int size\_L = Num.size();

int size\_R = Num2.Num.size();

if (!this->isminus && !Num2.isminus)

{

if (size\_L > size\_R)

return 1;

else if (size\_L < size\_R)

return 0;

else

{

for (int i = size\_L - 1; i >= 0; i--)

{

if (this->Num[i] - Num2.Num[i] > 0)

return 1;

else if (this->Num[i] - Num2.Num[i] < 0)

return 0;

else continue;

}

return 0;

}

}

else if (this->isminus && Num2.isminus)

{

if (size\_L > size\_R)

return 0;

else if (size\_L < size\_R)

return 1;

else

{

for (int i = size\_L - 1; i >= 0; i--)

{

if (this->Num[i] - Num2.Num[i] > 0)

return 0;

else if (this->Num[i] - Num2.Num[i] < 0)

return 1;

else continue;

}

return 0;

}

}

else if (!this->isminus && Num2.isminus)

return 1;

else return 0;

}

bool operator<(const Int& Num2) const

{

return (!(\*this > Num2));

}

bool operator==(const Int& Num2) const

{

if (this->isminus == Num2.isminus)

{

if (this->Num == Num2.Num)

return 1;

}

else return 0;

}

bool operator!=(const Int& Num2) const

{

return (!(\*this == Num2));

}

Int operator+(const Int& Num2) const

{

Int res;

int temp = 0;

const int size\_L = Num.size(), size\_R = Num2.Num.size();

bool LorR = 0;

if (size\_L >= size\_R)

LorR = 1;

if (LorR) res.Num.resize(size\_L + 1);

else res.Num.resize(size\_R + 1);

if (this->isminus && Num2.isminus)

res.isminus = 1,res.isskip = 1;

else if (!this->isminus && !Num2.isminus);

else if (!this->isminus && Num2.isminus)

{

res = Num2;

res.isminus = !res.isminus;

if (\*this == res)

{

Int res2;

res2.Num.emplace\_back(0);

return res2;

}

if (\*this > res)

{

res = (\*this - res);

res.isminus = 0;

return res;

}

else if (\*this < res)

{

res = (\*this - res);

res.isminus = 1;

res.isskip = 1;

return res;

}

}

else if (this->isminus && !Num2.isminus)

{

res = \*this;

res.isminus = !res.isminus;

if (res == Num2)

{

Int res2;

res2.Num.emplace\_back(0);

return res2;

}

if (res < Num2)

{

res = (res - Num2);

res.isminus = 0;

return res;

}

else if (res > Num2)

{

res = (Num2 - res);

res.isminus = 1;

res.isskip = 1;

return res;

}

}

for (int i = 0; i < min(size\_L,size\_R); i++)

{

temp = this->Num[i] + Num2.Num[i] + res.Num[i];

if (temp >= base)

{

temp -= base;

res.Num[i + 1] += 1;

}

res.Num[i] += temp - res.Num[i];

}

if(LorR)

for (int i = min(size\_L, size\_R); i < max(size\_L, size\_R); i++)

{

temp = this->Num[i] + res.Num[i];

if (temp >= base)

{

temp -= base;

res.Num[i + 1] += 1;

}

res.Num[i] += temp - res.Num[i];

}

else

for (int i = min(size\_L, size\_R); i < max(size\_L, size\_R); i++)

{

temp = Num2.Num[i] + res.Num[i];

if (temp >= base)

{

temp -= base;

res.Num[i + 1] += 1;

}

res.Num[i] += temp - res.Num[i];

}

if (res.Num[res.Num.size() - 1] == 0)

res.Num.resize(res.Num.size() - 1);

return res;

}

Int operator-(const Int& Num2) const

{

Int res;

if (this->isminus && Num2.isminus)

{

if (\*this > Num2)

{

res = ((-Num2) - (-\*this));

return res;

}

else if (\*this < Num2)

{

res = ((-\*this) - (-Num2));

return res;

}

else

{

res.Num.emplace\_back(0);

return res;

}

}

else if (!this->isminus && !Num2.isminus);

else if (!this->isminus && Num2.isminus)

{

return (\*this + (-Num2));

}

else if (this->isminus && !Num2.isminus)

{

return ((-Num2) + \*this);

}

if(\*this == Num2)

{

res.Num.emplace\_back(0);

return res;

}

else if (\*this > Num2)

{

int size\_L = this->Num.size();

int size\_R = Num2.Num.size();

int temp = 0;

res = \*this;

for (int i = size\_R - 1; i >= 0; i--)

{

temp = res.Num[i] - Num2.Num[i];

if (temp < 0)

{

res.Num[i + 1] -= 1;

res.Num[i] += base;

}

res.Num[i] -= Num2.Num[i];

}

for (int i = res.Num.size() - 1, n = 0; i >= 0; i--,n++)

{

if (res.Num[i] != 0)

{

res.Num.resize(res.Num.size() - n);

break;

}

}

return res;

}

else if (\*this < Num2)

{

res = (Num2 - \*this);

res.isminus = 1, res.isskip = 1;

return res;

}

}

Int operator\*(const Int& Num2) const

{

Int res;

if (Num2.isminus != this->isminus)

res.isminus = 1, res.isskip = 1;

else;

int size\_L = this->Num.size();

int size\_R = Num2.Num.size();

res.Num.resize((size\_L + size\_R) \* 4);

bool LorR = 0;

int temp = 0;

for (int i = 0; i < size\_L; i++)

{

for (int j = 0; j < size\_R; j++)

{

temp = Num[i] \* Num2.Num[j];

res.Num[i + j] += temp % base;

while (temp >= base)

{

res.Num[i + j + 1]++;

temp -= base;

}

if (res.Num[i + j] >= base)

{

res.Num[i + j + 1]++;

res.Num[i + j] -= base;

}

}

}

int cnt = 0;

for (int i = res.Num.size() - 1; i >= 0; i--, cnt++)

if (res.Num[i] != 0) break;

res.Num.resize(res.Num.size() - cnt);

return res;

}

Int operator/(const int num) const

{

Int res;

int size\_L = this->Num.size();

res.Num.resize(size\_L);

int x = 0;

for (int i = size\_L - 1; i >= 0; i--)

{

res.Num[i] = (x \* base + this->Num[i]) / num;

x = (x \* base + this->Num[i]) % num;

}

for (int i = size\_L - 1; i >= 0; i--)

{

if (res.Num[i] != 0) break;

if (res.Num[i] == 0) res.Num.pop\_back();

}

return res;

}

Int operator/(const Int& num) //减法模拟

{

if (\*this < num) return Int("0");

stringstream ss;

string temp;

string s1;

string s2;

string res;

for (int i = this->Num.size() - 1; i >= 0; i--)

{

ss << this->Num[i];

ss >> temp;

s1 += temp;

ss.clear();

}

for (int i = num.Num.size() - 1; i >= 0; i--)

{

ss << num.Num[i];

ss >> temp;

s2 += temp;

ss.clear();

}

temp = s2;

int cnt = s1.size() - s2.size() + 1;

res.resize(cnt,'0');

int count = 0;

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

{

move(s2, i + count);

while (s1 >= s2 && s1.size() == s2.size())

{

res[i] += 1;

s1 = minuss(s1, s2);

}

for(int j = 0;j < s1.size();j++)

if (s1[j] == '0' && s2[j] == '0') i++;

if (s1[0] != '0' && s1 < s2)

{

s2 = "0";

s2 += temp;

}

else if(s1[0] == '0')s2 = temp;

i -= validize(s1);

i += 1;

}

string res2;

/\*if (res[0] == '0')

{

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

res2 += res[i];

return Int(res2);

}\*/if (res[cnt - 1] == '0') res.pop\_back();

/\*else\*/ reverse(res.begin(), res.end());

return Int(res);

}

void move(string& target,int i)

{

for (; i > 0; i--)

target += '0';

}

void input()

{

string temp;

cin >> temp;

static int cnt = 0;

if (temp[0] != '-' && (temp[0] < '0' || temp[0] > '9'))

{

Error:

printf("Error Input!\n");

if (cnt == 1) {cnt = 0; return;}

else { cnt++; input(); }

}

else

{

regex pattern("^([1-9]|-)([0-9]\*)");

cmatch res;

if (regex\_match(temp.c\_str(), res, pattern))

{

Int n(temp);

\*this = n;

}

else goto Error;

}

Int n(temp);

\*this = n;

}

void output() const

{

int size = Num.size();

if (size == 0) cout << "Empty number\n";

if (isminus && isskip)

cout << "-";

printf("%d", Num[size - 1]);

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

printf("%.4d", Num[i]);

}

};

istream& operator>>(istream& cin, Int& Num)

{

Num.input();

return cin;

}

ostream& operator<<(ostream& cout,const Int& Num)

{

Num.output();

return cout;

}

Int operator-(const Int& Num)

{

Int res = Num;

res.isminus = !res.isminus;

return res;

}

int main()

{

while (1)

{

Int num1;

int n = 0;

cin >> num1 >> n;

cout << num1 / n << endl << endl;

}

}

（VS2010 非法警告删除）

#include<iostream>

#include<vector>

#include<sstream>

#include<string>

using namespace std;

const int base = 10000;

bool isBorrow(int i, string& s1, string& s2)

{

return s1[i] - s2[i] < 0;

}

int validize(string& s)

{

int cnt = 0;

for (int i = 0; i < s.size(); i++)

{

if (s[i] == '0') cnt++;

if (s[i] != '0') break;

}

if (cnt == s.size()) return 0;

if (!cnt) return 1;

else

{

string temp;

for (int i = cnt; i < s.size(); i++)

temp += s[i];

s = temp;

}

return cnt;

}

string minuss(string& s1, string& s2)

{

int len = s1.size();

string res;

string temp1 = s1;

string temp2 = s2;

if (s1 < s2)

{

res += "-1";

return res;

}

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

{

again:

if (isBorrow(i, temp1, temp2))

{

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

{

if (temp1[j] > '0')

{

temp1[j] -= 1;

temp1[j + 1] += 10;

break;

}

}

if (isBorrow(i, temp1, temp2))

goto again;

}

temp1[i] -= temp2[i];

temp1[i] += 48;

}

res = temp1;

return res;

}

class Int

{

friend ostream& operator<<(ostream& cout, const Int& Num);

friend Int operator-(const Int& Num);

friend istream& operator>>(istream& cin, Int& Num);

vector<int> Num;

bool isminus;

bool isskip;

public:

Int():isminus(0),isskip(0) {};

Int(string &s):isminus(0),isskip(0)

{

int len = s.size();

stringstream ss;

int temp = 0;

if (s[0] == '-')

isminus = 1;

while (len / 4)

{

ss << s.substr(len - 3 - 1, 4);

ss >> temp;

ss.clear();

Num.emplace\_back(temp);

len -= 4;

}

if (len)

{

int i = 0;

if (isminus) isskip = 1;

if (isskip) i = 1, len -= 1;

ss << s.substr(i, len);

ss >> temp;

ss.clear();

Num.emplace\_back(temp);

}

else if (isminus && !len)

Num[Num.size() - 1] = -Num[Num.size() - 1];

}

Int(const char\* c)

{

string s = c;

Int temp(s);

\*this = temp;

}

bool operator>(const Int& Num2) const

{

int size\_L = Num.size();

int size\_R = Num2.Num.size();

if (!this->isminus && !Num2.isminus)

{

if (size\_L > size\_R)

return 1;

else if (size\_L < size\_R)

return 0;

else

{

for (int i = size\_L - 1; i >= 0; i--)

{

if (this->Num[i] - Num2.Num[i] > 0)

return 1;

else if (this->Num[i] - Num2.Num[i] < 0)

return 0;

else continue;

}

return 0;

}

}

else if (this->isminus && Num2.isminus)

{

if (size\_L > size\_R)

return 0;

else if (size\_L < size\_R)

return 1;

else

{

for (int i = size\_L - 1; i >= 0; i--)

{

if (this->Num[i] - Num2.Num[i] > 0)

return 0;

else if (this->Num[i] - Num2.Num[i] < 0)

return 1;

else continue;

}

return 0;

}

}

else if (!this->isminus && Num2.isminus)

return 1;

else return 0;

}

bool operator<(const Int& Num2) const

{

return (!(\*this > Num2));

}

bool operator==(const Int& Num2) const

{

if (this->isminus == Num2.isminus)

{

if (this->Num == Num2.Num)

return 1;

}

else return 0;

}

bool operator!=(const Int& Num2) const

{

return (!(\*this == Num2));

}

Int operator+(const Int& Num2) const

{

Int res;

int temp = 0;

const int size\_L = Num.size(), size\_R = Num2.Num.size();

bool LorR = 0;

if (size\_L >= size\_R)

LorR = 1;

if (LorR) res.Num.resize(size\_L + 1);

else res.Num.resize(size\_R + 1);

if (this->isminus && Num2.isminus)

res.isminus = 1,res.isskip = 1;

else if (!this->isminus && !Num2.isminus);

else if (!this->isminus && Num2.isminus)

{

res = Num2;

res.isminus = !res.isminus;

if (\*this == res)

{

Int res2;

res2.Num.emplace\_back(0);

return res2;

}

if (\*this > res)

{

res = (\*this - res);

res.isminus = 0;

return res;

}

else if (\*this < res)

{

res = (\*this - res);

res.isminus = 1;

res.isskip = 1;

return res;

}

}

else if (this->isminus && !Num2.isminus)

{

res = \*this;

res.isminus = !res.isminus;

if (res == Num2)

{

Int res2;

res2.Num.emplace\_back(0);

return res2;

}

if (res < Num2)

{

res = (res - Num2);

res.isminus = 0;

return res;

}

else if (res > Num2)

{

res = (Num2 - res);

res.isminus = 1;

res.isskip = 1;

return res;

}

}

for (int i = 0; i < min(size\_L,size\_R); i++)

{

temp = this->Num[i] + Num2.Num[i] + res.Num[i];

if (temp >= base)

{

temp -= base;

res.Num[i + 1] += 1;

}

res.Num[i] += temp - res.Num[i];

}

if(LorR)

for (int i = min(size\_L, size\_R); i < max(size\_L, size\_R); i++)

{

temp = this->Num[i] + res.Num[i];

if (temp >= base)

{

temp -= base;

res.Num[i + 1] += 1;

}

res.Num[i] += temp - res.Num[i];

}

else

for (int i = min(size\_L, size\_R); i < max(size\_L, size\_R); i++)

{

temp = Num2.Num[i] + res.Num[i];

if (temp >= base)

{

temp -= base;

res.Num[i + 1] += 1;

}

res.Num[i] += temp - res.Num[i];

}

if (res.Num[res.Num.size() - 1] == 0)

res.Num.resize(res.Num.size() - 1);

return res;

}

Int operator-(const Int& Num2) const

{

Int res;

if (this->isminus && Num2.isminus)

{

if (\*this > Num2)

{

res = ((-Num2) - (-\*this));

return res;

}

else if (\*this < Num2)

{

res = ((-\*this) - (-Num2));

return res;

}

else

{

res.Num.emplace\_back(0);

return res;

}

}

else if (!this->isminus && !Num2.isminus);

else if (!this->isminus && Num2.isminus)

{

return (\*this + (-Num2));

}

else if (this->isminus && !Num2.isminus)

{

return ((-Num2) + \*this);

}

if(\*this == Num2)

{

res.Num.emplace\_back(0);

return res;

}

else if (\*this > Num2)

{

int size\_L = this->Num.size();

int size\_R = Num2.Num.size();

int temp = 0;

res = \*this;

for (int i = size\_R - 1; i >= 0; i--)

{

temp = res.Num[i] - Num2.Num[i];

if (temp < 0)

{

res.Num[i + 1] -= 1;

res.Num[i] += base;

}

res.Num[i] -= Num2.Num[i];

}

for (int i = res.Num.size() - 1, n = 0; i >= 0; i--,n++)

{

if (res.Num[i] != 0)

{

res.Num.resize(res.Num.size() - n);

break;

}

}

return res;

}

else if (\*this < Num2)

{

res = (Num2 - \*this);

res.isminus = 1, res.isskip = 1;

return res;

}

}

Int operator\*(const Int& Num2) const

{

Int res;

if (Num2.isminus != this->isminus)

res.isminus = 1, res.isskip = 1;

else;

int size\_L = this->Num.size();

int size\_R = Num2.Num.size();

res.Num.resize((size\_L + size\_R) \* 4);

bool LorR = 0;

int temp = 0;

for (int i = 0; i < size\_L; i++)

{

for (int j = 0; j < size\_R; j++)

{

temp = Num[i] \* Num2.Num[j];

res.Num[i + j] += temp % base;

while (temp >= base)

{

res.Num[i + j + 1]++;

temp -= base;

}

if (res.Num[i + j] >= base)

{

res.Num[i + j + 1]++;

res.Num[i + j] -= base;

}

}

}

int cnt = 0;

for (int i = res.Num.size() - 1; i >= 0; i--, cnt++)

if (res.Num[i] != 0) break;

res.Num.resize(res.Num.size() - cnt);

return res;

}

Int operator/(const int num) const

{

Int res;

int size\_L = this->Num.size();

res.Num.resize(size\_L);

int x = 0;

for (int i = size\_L - 1; i >= 0; i--)

{

res.Num[i] = (x \* base + this->Num[i]) / num;

x = (x \* base + this->Num[i]) % num;

}

for (int i = size\_L - 1; i >= 0; i--)

{

if (res.Num[i] != 0) break;

if (res.Num[i] == 0) res.Num.pop\_back();

}

return res;

}

Int operator/(const Int& num) //减?法ぁ?模￡拟a

{

if (\*this < num) return Int("0");

stringstream ss;

string temp;

string s1;

string s2;

string res;

for (int i = this->Num.size() - 1; i >= 0; i--)

{

ss << this->Num[i];

ss >> temp;

s1 += temp;

ss.clear();

}

for (int i = num.Num.size() - 1; i >= 0; i--)

{

ss << num.Num[i];

ss >> temp;

s2 += temp;

ss.clear();

}

temp = s2;

int cnt = s1.size() - s2.size() + 1;

res.resize(cnt,'0');

int count = 0;

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

{

move(s2, i + count);

while (s1 >= s2 && s1.size() == s2.size())

{

res[i] += 1;

s1 = minuss(s1, s2);

}

for(int j = 0;j < s1.size();j++)

if (s1[j] == '0' && s2[j] == '0') i++;

if (s1[0] != '0' && s1 < s2)

{

s2 = "0";

s2 += temp;

}

else if(s1[0] == '0')s2 = temp;

i -= validize(s1);

i += 1;

}

string res2;

/\*if (res[0] == '0')

{

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

res2 += res[i];

return Int(res2);

}\*/if (res[cnt - 1] == '0') res.pop\_back();

/\*else\*/ reverse(res.begin(), res.end());

return Int(res);

}

void move(string& target,int i)

{

for (; i > 0; i--)

target += '0';

}

void input()

{

string temp;

cin >> temp;

static int cnt = 0;

Int n(temp);

\*this = n;

}

void output() const

{

int size = Num.size();

if (size == 0) cout << "Empty number\n";

if (isminus && isskip)

cout << "-";

printf("%d", Num[size - 1]);

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

printf("%.4d", Num[i]);

}

};

istream& operator>>(istream& cin, Int& Num)

{

Num.input();

return cin;

}

ostream& operator<<(ostream& cout,const Int& Num)

{

Num.output();

return cout;

}

Int operator-(const Int& Num)

{

Int res = Num;

res.isminus = !res.isminus;

return res;

}

int main()

{

while (1)

{

Int num1;

int n = 0;

cin >> num1 >> n;

cout << num1 / n << endl << endl;

}

}

问题与思考：

原本用了getline去读取，但是和原生类型输入放一起的时候出现了问题。需要统一输入的接口，防止出现问题。统一用cin>>不能读取空格（本题不应该有），若getline则后者输入（int）n之后缓冲区还留着换行，导致下一次碰到getline直接跑路，num1直接变成默认空白构造（这里未提供初始值等默认信息）。



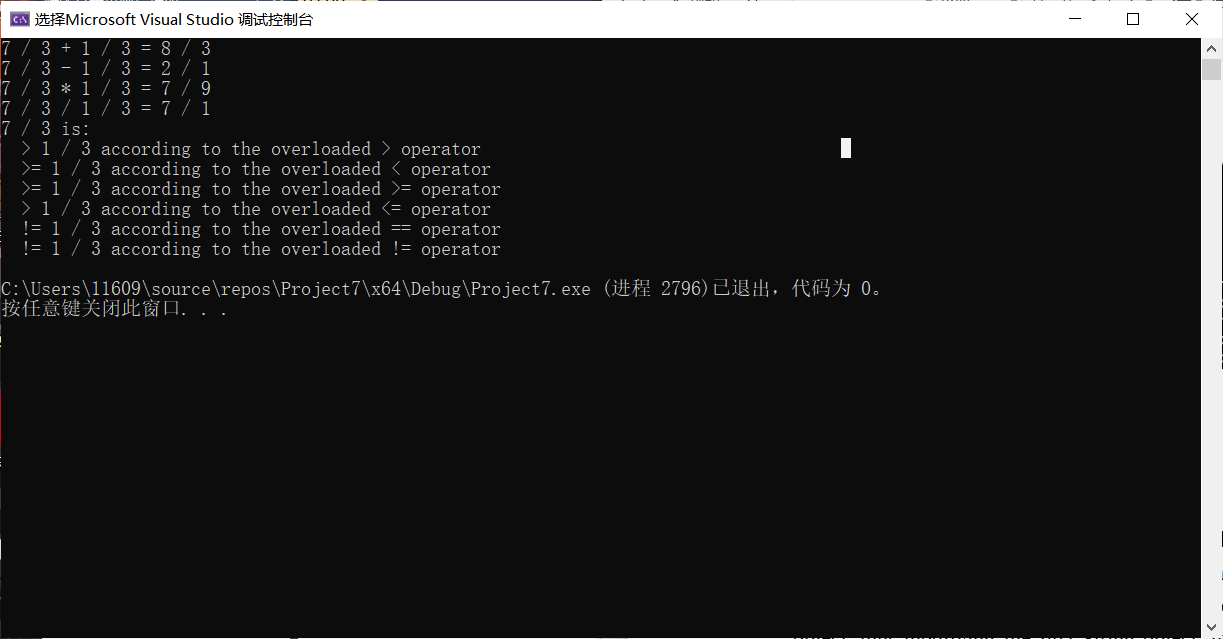
输入时对非法输入进行警告并要求再次输入，若错误次数过多取消正常构造，实际上去采用默认值会好些。

压位加减乘都挺顺利，除法的时候发现还是得回头一位一位操作除法。除法采用减法模拟，将位数扩充至统一后相减，以此往复，进行相减，每一位上最多进行10次减法运算。

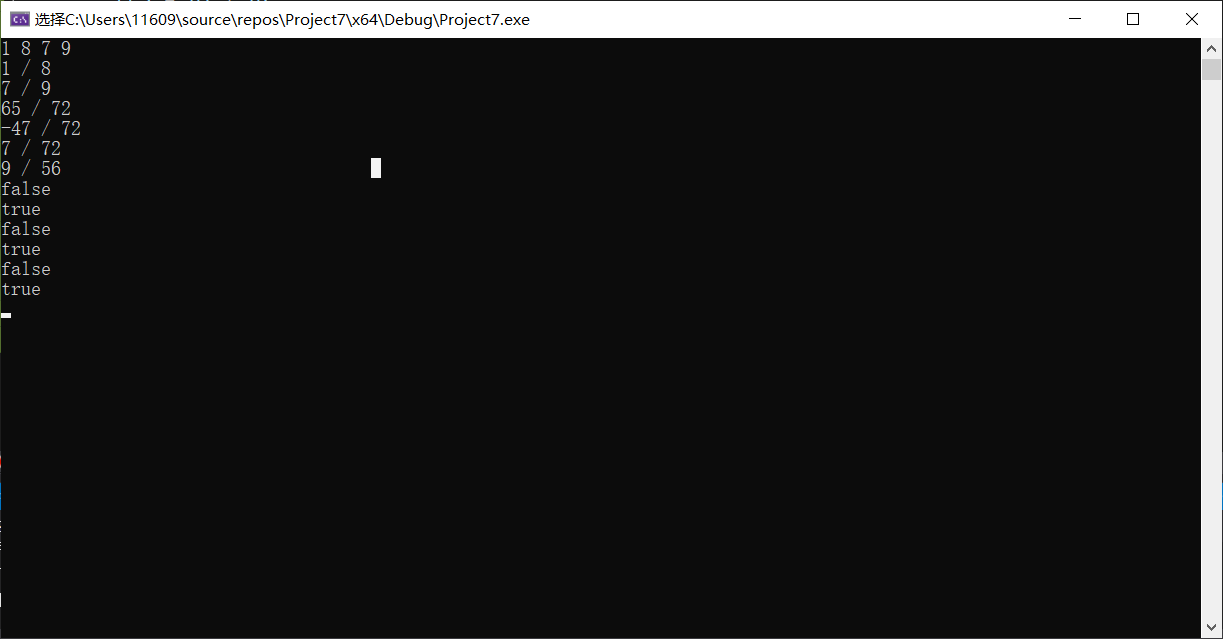
内存上直接开大预设会方便特别多。

**EX3: Rational Number Class**

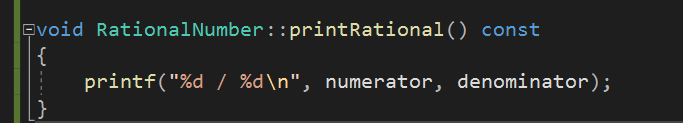
题中所给测试代码效果：



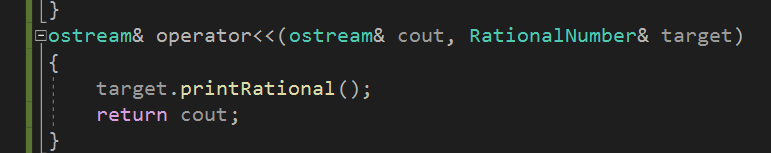
自行测试效果：



希望以如下格式进行输出



所以重载的<< 直接调用printRational()



代码：

头文件：

// Exercise 11.15 Solution: RationalNumber.h

// RationalNumber class definition.

#include <iostream>

using namespace std; //不在这加上这两，在下方重载cin，编译器会爆出奇怪的错误

#ifndef RATIONAL\_NUMBER\_H

#define RATIONAL\_NUMBER\_H

//也可#pragma once

class RationalNumber

{

friend istream& operator>>(istream& cin,RationalNumber& target);

friend ostream& operator<<(ostream& cout, RationalNumber& target);

public:

RationalNumber(int = 0, int = 1); // default constructor //两个参数且均有默认值，explicit似乎失效

RationalNumber operator+(const RationalNumber&); // addition

RationalNumber operator-(const RationalNumber&) const; // subtraction

RationalNumber operator\*(const RationalNumber&); // multiplication

RationalNumber operator/(RationalNumber&); // division

// relational operators

bool operator>(const RationalNumber&) const;

bool operator<(const RationalNumber&) const;

bool operator>=(const RationalNumber&) const;

bool operator<=(const RationalNumber&) const;

// equality operators

bool operator==(const RationalNumber&) const;

bool operator!=(const RationalNumber&) const;

void printRational() const; // display rational number

private:

int numerator; // private variable numerator分子

int denominator; // private variable denominator分母

void reduction(); // function for fraction reduction化简

}; // end class RationalNumber

istream& operator>>(istream& cin, RationalNumber& target); //便于调试数据

ostream& operator<<(ostream& cout, RationalNumber& target);

#endif

实现文件：

#include"RATIONAL\_NUMBER\_H.h"

#include<conio.h>

void RationalNumber::reduction()

{

bool isPositive = 1;

if (numerator == 0) return;

else if (numerator < 0)

{

numerator = -numerator;

isPositive = 0;

}

int a = numerator, b = denominator;

int temp = 0;

if (a < b) swap(a, b);

while (b != 0) //欧几里得

{

temp = a;

a = b;

b = temp % b;

}

//while (a != b) //更相减损术

//{

// if (a > b) a = a - b;

// else b = b - a;

//}

if (isPositive == 0) numerator = -numerator;

numerator /= a;

denominator /= a;

}

RationalNumber::RationalNumber(int nume, int deno)

{

if (deno > 0)

{

numerator = nume;

denominator = deno;

reduction();

}

else

{

printf("Error Input\n");

numerator = 0;

denominator = 1;

}

}

RationalNumber RationalNumber::operator+(const RationalNumber& target)

{

if (denominator == target.denominator)

{

RationalNumber res(numerator + target.numerator, denominator);

res.reduction();

return res;

}

else

{

RationalNumber res(numerator \* target.denominator + target.numerator \* denominator, denominator \* target.denominator);

res.reduction();

return res;

}

}

istream& operator>>(istream& cin, RationalNumber& target)

{

int nume,deno;

cin >> nume >> deno;

RationalNumber temp(nume,deno);

target = temp;

return cin; //需要考虑合法性，不可直接访问数据，否则下方若进行违法输入则"Error Input"的信息只出现了一次。

}

ostream& operator<<(ostream& cout, RationalNumber& target)

{

target.printRational();

return cout;

}

RationalNumber RationalNumber::operator-(const RationalNumber& target) const

{

if (denominator == target.denominator)

{

RationalNumber res(numerator - target.numerator,denominator);

res.reduction();

return res;

}

else

{

RationalNumber res(numerator \* target.denominator - target.numerator \* denominator, denominator \* target.denominator);

res.reduction();

return res;

}

}

RationalNumber RationalNumber::operator\*(const RationalNumber& target)

{

RationalNumber res(numerator \* target.numerator, denominator \* target.denominator);

res.reduction();

return res;

}

RationalNumber RationalNumber::operator/(RationalNumber& target)

{

if (target.numerator == 0)

{

printf("Error Operation\n");

return target;

}

RationalNumber temp(target.denominator, target.numerator);

RationalNumber res = \*this \* temp;

return res;

}

bool RationalNumber::operator>(const RationalNumber& target) const

{

RationalNumber res = \*this - target; // > 用了const常函数,若希望调用-则同样需要补上const

if (res.numerator > 0) return 1; //其他的头文件中没给出const，这里不进行修改，原则上都应前后两个const

else return 0;

}

bool RationalNumber::operator<(const RationalNumber& target) const

{

return (!(\*this > target));

}

bool RationalNumber::operator>=(const RationalNumber& target) const

{

RationalNumber res = \*this - target;

if (res.numerator >= 0) return 1;

else return 0; //这一段不知道有没有复用代码的办法；

}

bool RationalNumber::operator<=(const RationalNumber& target) const

{

RationalNumber res = \*this - target; //统一调用-如果出错容易查

if (res.numerator <= 0) return 1;

else return 0;

}

// equality operators

bool RationalNumber::operator==(const RationalNumber& target) const

{

RationalNumber res = \*this - target;

if (res.numerator == 0) return 1;

else return 0;

}

bool RationalNumber::operator!=(const RationalNumber& target) const

{

return (!(\*this == target));

}

void RationalNumber::printRational() const

{

printf("%d / %d\n", numerator, denominator);

}

int main()

{

while (1)

{

RationalNumber n1, n2, res;

cin >> n1 >> n2;

res = n1 + n2;

cout << n1 << n2 << res;

res = n1 - n2;

cout << res;

res = n1 \* n2;

cout << res;

res = n1 / n2;

cout << res;

cout << boolalpha << (n1 > n2) << endl; //这里<<运算符优先级大于>等运算符，故加上括号

cout << boolalpha << (n1 < n2) << endl;

cout << boolalpha << (n1 >= n2) << endl;

cout << boolalpha << (n1 <= n2) << endl;

cout << boolalpha << (n1 == n2) << endl;

cout << boolalpha << (n1 != n2) << endl;

\_getch();

}

/\*RationalNumber n1('/', '/');

cout << n1; //这里并不清楚如何避免非法的符号输入，如此处

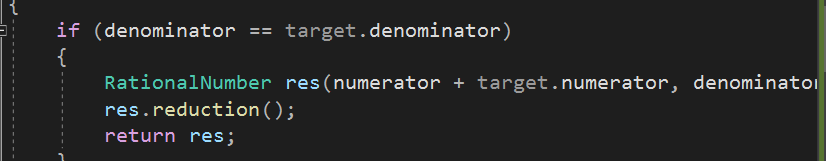
发生char转int，从传进去就是int了\*/

}

问题与思考：

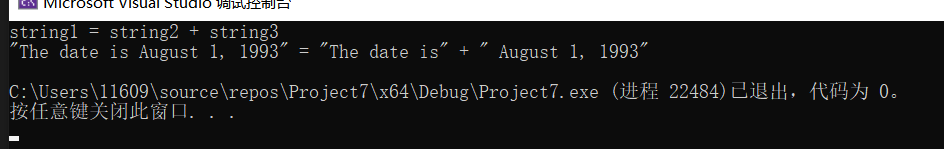
1. 代码复用是极为舒适的，除了快速解决其他功能这一点，倘若其他的功能均出现了问题(比如此题比较大小相等与不等关系均调用减法)，很有可能就是公共调用的方法出现问题，可以快速定位到错误的函数；
2. explicit可以避免一些隐式的类型转换，但是对于函数参数有一定要求。本题中如上方注释提到的非法符号输入不知如何解决，‘/’这样的符号会直接由char转换为int（算数类型转换？），并不知道如何避免这样的转换；
3. 上方括号原因是运算符优先级问题，<<运算符优先级是大于>的；
4. 题中如下图这些地方直接用表达式结果对结果对象进行了构造，减少内存读取，不知这里是否有较大意义。

5.构造函数中也可以return结束函数，但是应该准备默认值，防止创造出一个没有初始化的对象。

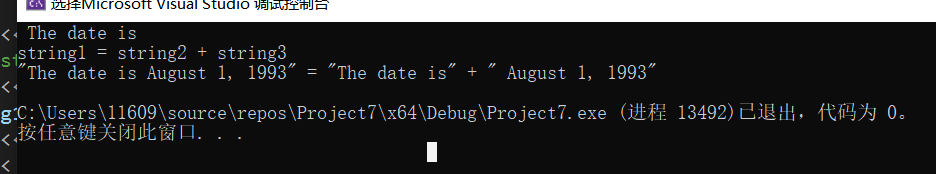


**EX4: String Concatenation**

题中测试代码效果：



测试空string：



//#define \_CRT\_SECURE\_NO\_WARNINGS vs2019解决\_s等的宏定义

#include <iostream>

#include <cstring>

#include <cassert>

#include"String.h"

#include<cstdlib>

using namespace std;

String::String(const char\* const str)

{

this->length = strlen(str);//实际应是size\_t

if (this->length == 0) //为空，放在栈里，不担心深浅拷贝

{

this->sPtr = (char\*)str;

return;

}

this->sPtr = new char[this->length + 5]; //如果长度够大可能算数溢出

char\* b = sPtr;

for (const char\* c = str; \*c != 0; c++, b++)

\*b = \*c;

\*b = 0;

}

//stl中的string长度应是size\_type(unsigned int/unsigned long)类型而非int

//问题与思考中会再补充

//可以String() = default;恢复默认构造

String::String(const String& str)

{

this->length = str.length;

if (this->length == 0)

{

this->sPtr = str.sPtr;

return;

}

this->sPtr = new char[this->length + 5]; //如果要重载下标访问需要再考虑一下合法性问题

char\* b = sPtr;

for (const char\* c = str.sPtr; \*c != 0; c++, b++)

\*b = \*c;

\*b = 0;

}

String::~String()

{

if (this->length != 0)

{

delete[](this->sPtr);

this->sPtr = NULL;

}

else return;

};

const String& String::operator=(const String& str)

{

this->length = str.length;

if (this->length == 0)

{

this->sPtr = str.sPtr;

return \*this;

}

else

{

this->sPtr = new char[this->length + 5];

char\* b = this->sPtr;

for (const char\* c = str.sPtr; \*c != 0; c++, b++)

\*b = \*c;

\*b = 0;

}

return \*this;

}

String String::operator+(const String& str)

{

String res;

res.length = this->length + str.length;

res.sPtr = new char[res.length + 5];

char\* s = res.sPtr;

for (const char\* b = this->sPtr; \*b != 0; b++, s++)

\*s = \*b;

for (const char\* c = str.sPtr; \*c != 0; c++, s++)

\*s = \*c;

\*s = 0;

return res;

}

ostream& operator<<(ostream& output, const String& s)

{

output << s.sPtr;

return output;

}

int main()

{

String string1, string2("The date is");

String string3(" August 1, 1993");

// test overloaded operators

cout << "string1 = string2 + string3\n";

string1 = string2 + string3; // tests overloaded = and + operator

cout << "\"" << string1 << "\" = \"" << string2 << "\" + \""

<< string3 << "\"" << endl;

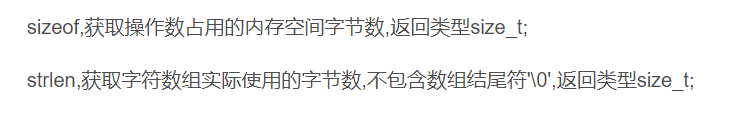
return 0;

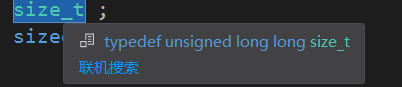
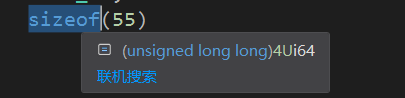
}

问题与思考：STL中采用的类型为size\_type, C标准库（cstdlib）中也有size\_t,比如sizeof（）以及strlen（）返回的就是size\_t，采用当前机器上最大非符号类型（下方就是unsigned long long），还有sizeof（）并不对放入其中的对象进行操作，往里面放\*野指针也不会有问题。

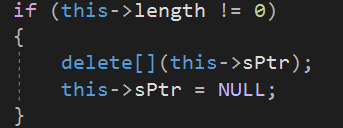
题目中给的长度是int，这里并不进行修改，但更应采用无符号类型，避免有符号与无符号转换（毕竟长度不可小于0）



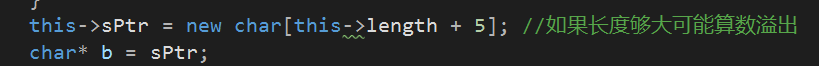




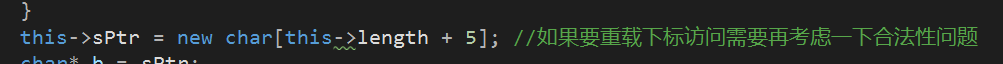
NULL其实可以初始化的不只有指针，也可以用于int等类型，C++11 推荐使用nullptr专门用于指针。



就题目而言，一些安全问题可以如下解决



改用无符号



需要进行初始化

还有一点课上也用到下方这个类型转换

