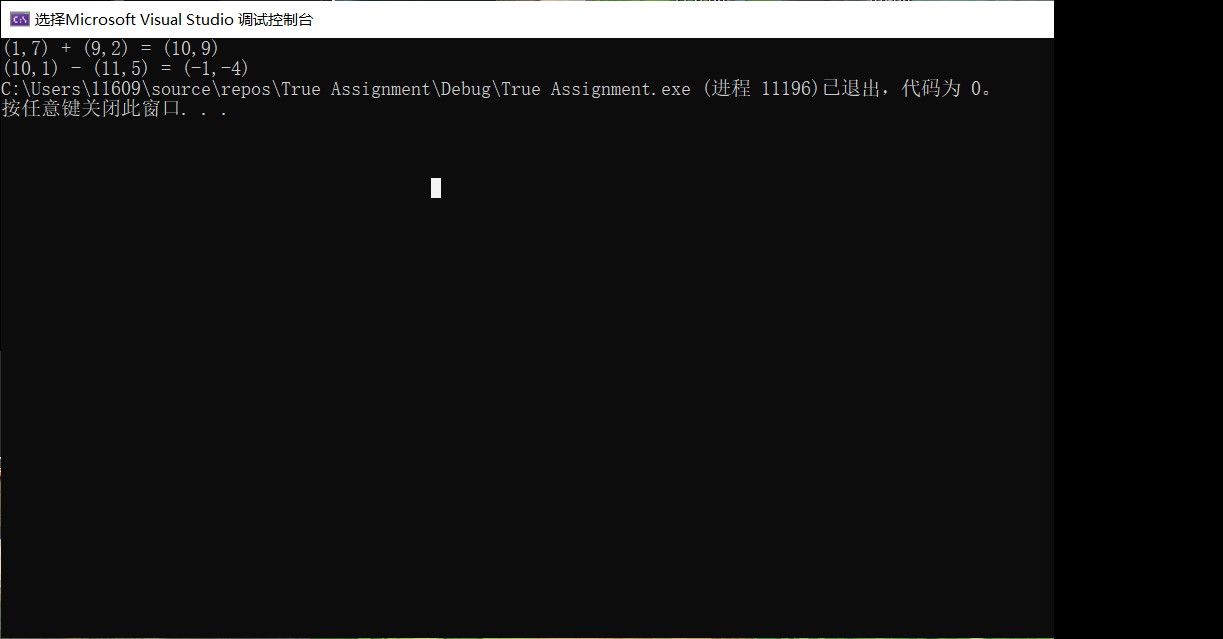
**EX1:( 9.5 Complex Class)**



#include<iostream>

#include<cstdio>

using namespace std;

class Complex

{

double real\_n;

double imag\_n;//(C++11)也可在这初始化或使用初始化列表，在这初始化可以解决警告。

public:

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

Complex Adding(Complex data)

{

Complex temp;

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

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

return temp;

} //值返回，仅用于返回计算结果

Complex Subtracting(Complex data)

{

Complex temp;

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

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

return temp;

}

void Printing()

{

cout << "(" << this->real\_n << "," << this->imag\_n << ")";

/\*printf("(%.0lf,%.0lf)", this->real\_n, this->imag\_n);\*/

//printf 需自行调整输出位数，cout可以输出有数据的位数

}

};

int main()

{

Complex n5;

Complex n1(1,7), n2(9,2);

Complex res = n1.Adding(n2);

n1.Printing();

cout << " + ";

n2.Printing();

cout << " = ";

res.Printing();

cout << endl;

Complex n3(10, 1), n4(11, 5);

Complex res2 = n3.Subtracting(n4);

n3.Printing();

cout << " - ";

n4.Printing();

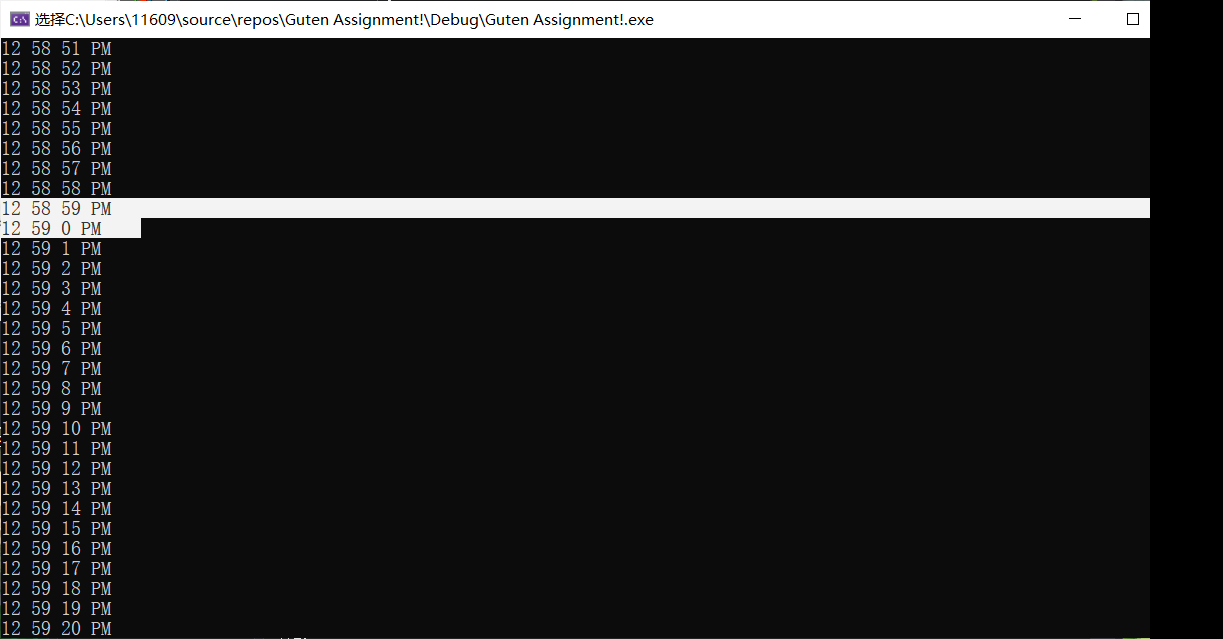
cout << " = ";

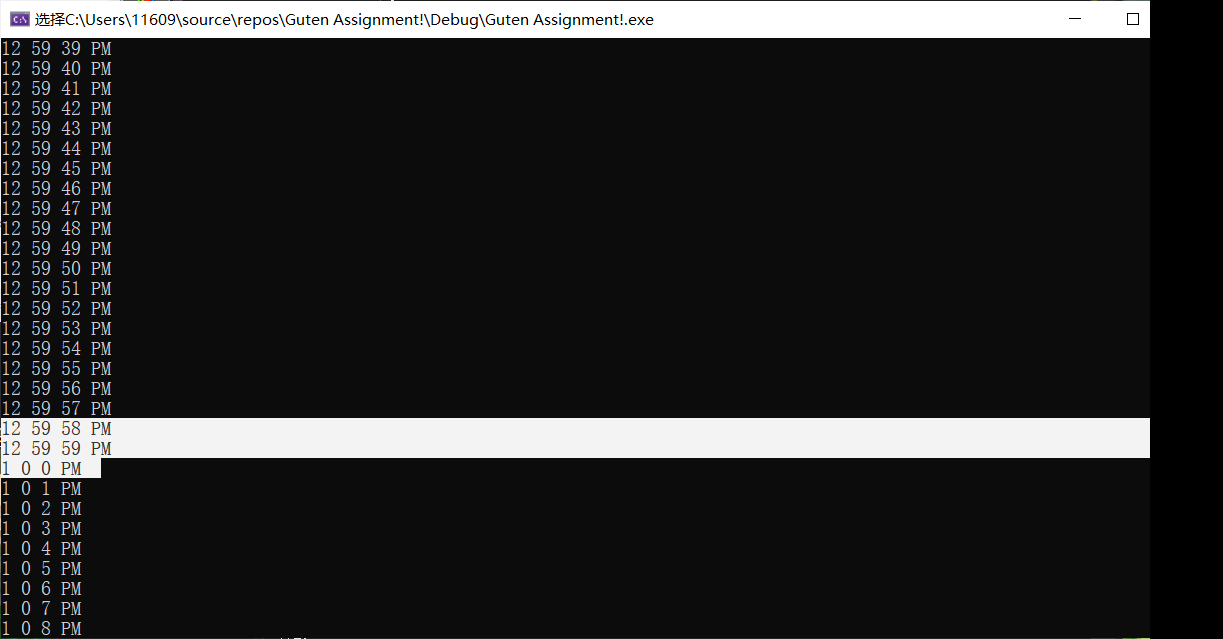
res2.Printing();

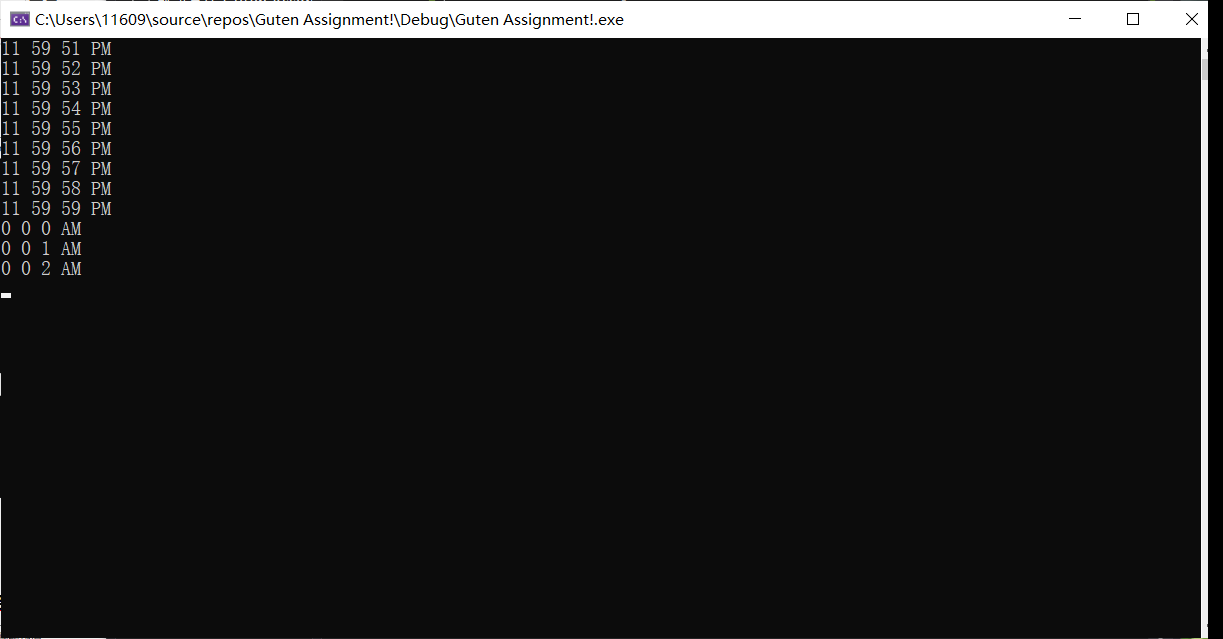
}

思考与发现：STL中其实有现成Complex类，是真的复数，还能用三角。

**EX2: (9.7, Enhancing Class Time)**







.h文件（题目给定）：

//#pragma once

#ifndef TIME\_H

#define TIME\_H

class Time

{

friend void tick(Time& t);

public:

explicit Time(int = 0, int = 0, int = 0);

//防止隐式转换

void setTime(int, int, int);

void setHour(int);

void setMinute(int);

void setSecond(int);

unsigned int getHour() const;

unsigned int getMinute() const;

unsigned int getSecond() const;

void printUniversal() const;

void printStandard() const;

private:

unsigned int hour;

unsigned int minute;

unsigned int second;

};

#endif

.cpp文件：

#include"TIME\_H.h"

#include<iostream>

#include<cstdio>

#include<conio.h>

using namespace std;

Time::Time(int hour, int minute, int second)

{

setTime(hour, minute, second);

}

void Time::setTime(int h, int m, int s)

{

setHour(h);

setMinute(m);

setSecond(s);

}

void Time::setHour(int h)

{

if (h >= 0 && h < 24)

hour = h;

else

throw invalid\_argument("hour must be 0-23");

}

void Time::setMinute(int m)

{

if (m >= 0 && m < 60)

minute = m;

else

throw invalid\_argument("minute must be 0-59");

}

void Time::setSecond(int s)

{

if (s >= 0 && s < 60)

second = s;

else

throw invalid\_argument("second must be 0-59");

}

unsigned int Time::getHour() const

{

return hour;

}

unsigned int Time::getMinute() const

{

return minute;

}

unsigned int Time::getSecond() const

{

return second;

}

void Time::printUniversal() const

{

printf("%u %u %u\n", hour, minute, second);

}

void Time::printStandard() const

{

if (hour > 12)

{

printf("%u %u %u PM\n", hour - 12, minute, second);

}

else if (hour < 12)

{

printf("%u %u %u AM\n", hour, minute, second);

}

else if (hour == 12)

{

printf("%u %u %u PM\n", hour, minute, second);

}

}

void tick(Time& t)

{

t.second++;

if (t.second > 59)

t.second = 0, t.minute++;

if (t.minute > 59)

t.minute = 0, t.hour++;

if (t.hour > 23)

t.hour = 0;

}

int main()

{

Time t(23,59,51);

while (1)

{

t.printStandard();

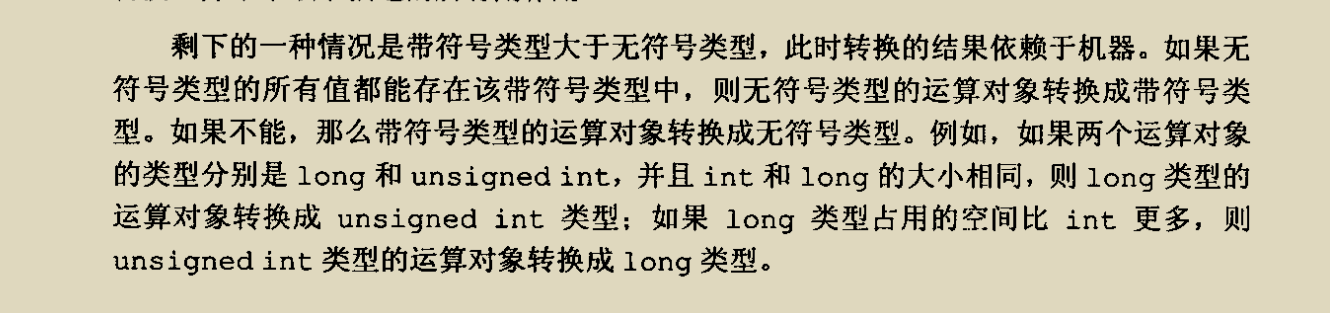
tick(t);

\_getch();

}

}

问题与思考：题目给定头文件中，构造函数加上了explicit防止隐式转换，这一点很有意思。题中数据成员均使用unsigned int，如果传递(int)-1进去，转为unsigned int即为-1%2^32,得到2^32-1,再转换成int直接超出数据范围，如果函数采用unsigned int传入，用户却输入（int）-1，此时的转换就会出现问题，如果是修改数据，一个带符号的数大于不带符号的数，不带符号的数可能转换为带符号的数。在《C++ primer》上见过：142页第五版



**EX3: (9.14 HugeInteger Class)**



.h文件使用题目给定

//也可使用#pragma once

#ifndef HUGEINTEGER\_H

#define HUGEINTEGER\_H

class HugeInteger

{

public:

HugeInteger(int = 0); // conversion/default constructor

HugeInteger(const char\*); // conversion constructor

// addition operator; HugeInteger + HugeInteger

HugeInteger add(const HugeInteger&);

// addition operator; HugeInteger + int

HugeInteger add(int);

// addition operator;

// HugeInteger + string that represents large integer value

HugeInteger add(const char\*);

// subtraction operator; HugeInteger - HugeInteger

HugeInteger subtract(const HugeInteger&);

// subtraction operator; HugeInteger - int I1.substract(I2) return \*this;

HugeInteger subtract(int);

// subtraction operator;

// HugeInteger - string that represents large integer value

HugeInteger subtract(const char\*);

bool isEqualTo(HugeInteger&); // is equal to

bool isNotEqualTo(HugeInteger&); // not equal to

bool isGreaterThan(HugeInteger&); // greater than

bool isLessThan(HugeInteger&); // less than

bool isGreaterThanOrEqualTo(HugeInteger&); // greater than

// or equal to

bool isLessThanOrEqualTo(HugeInteger&); // less than or equal

bool isZero(); // is zero

void input(const char\*); // input

void output(); // output

private:

int integer[40]; // 40 element array

};

#endif

.cpp文件（实现与测试）：

#include"HUGEINTEGER\_H.h"

#include<iostream>

using namespace std;

HugeInteger::HugeInteger(int num)

{

for (int i = 39; i >= 0; i--)

{

integer[i] = num % 10;

num /= 10;

}

}

HugeInteger::HugeInteger(const char\* num)

{

int cnt = 0;

for (; num[cnt] != 0; cnt++);

cnt--;

for (int i = 39; i >= 0; i--,cnt--)

{

if (cnt >= 0)

integer[i] = num[cnt] - '0';

else integer[i] = 0;

}

}

HugeInteger HugeInteger::add(const HugeInteger& another)

{

HugeInteger res;

for (int i = 39; i >= 0; i--)

{

if (this->integer[i] + another.integer[i] < 10)

res.integer[i] += this->integer[i] + another.integer[i];

else

{

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

res.integer[i] += this->integer[i] + another.integer[i] - 10;

}

}

return res;

}

HugeInteger HugeInteger::add(int num)

{

HugeInteger another(num);

return add(another);

}

HugeInteger HugeInteger::add(const char\* num)

{

HugeInteger another(num);

return add(another);

}

HugeInteger HugeInteger::subtract(const HugeInteger& another)

{

HugeInteger res;

for (int i = 39; i >= 0; i--)

{

if (this->integer[i] - another.integer[i] < 0)

{

again:

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

{

if (this->integer[j] > 0)

{

this->integer[j]--;

this->integer[j + 1] += 10;

break;

}

}

if (this->integer[i] - another.integer[i] < 0)

goto again;

}

res.integer[i] += this->integer[i] - another.integer[i];

}

return res;

}

HugeInteger HugeInteger::subtract(int num)

{

HugeInteger another(num);

return this->subtract(another);

}

HugeInteger HugeInteger::subtract(const char\* num)

{

HugeInteger another(num);

return this->subtract(another);

}

bool HugeInteger::isEqualTo(HugeInteger& another)

{

for (int i = 39; i >= 0; i--)

{

if (this->integer[i] != another.integer[i])

return 0;

}

return 1;

}

bool HugeInteger::isNotEqualTo(HugeInteger& another)

{

return (!isEqualTo(another));

}

bool HugeInteger::isGreaterThan(HugeInteger& another)

{

int size = 0,size2 = 0;

for (int i = 39; i >= 0; i--)

{

if (!this->integer[i] && this->integer[i + 1] && i + 1 < 40);

else size++;

if (!another.integer[i] && another.integer[i + 1] && i + 1 < 40);

else size2++;

}

if (size > size2)

return 1;

else if (size < size2)

return 0;

else

for (int i = 0; i < 40; i++)

{

if (this->integer[i] - another.integer[i] > 0)

return 1;

else if (this->integer[i] - another.integer[i] < 0)

return 0;

else continue;

}

return 0;

}

bool HugeInteger::isLessThan(HugeInteger& another)

{

return(!isGreaterThan(another));

}

bool HugeInteger::isGreaterThanOrEqualTo(HugeInteger& another)

{

return (isGreaterThan(another) || isEqualTo(another));

}

bool HugeInteger::isLessThanOrEqualTo(HugeInteger& another)

{

return (!isGreaterThan(another) || isEqualTo(another));

}

bool HugeInteger::isZero()

{

for (int i = 0; i < 39; i++)

{

if (this->integer[i] != 0)

return 0;

}

return 1;

}

void HugeInteger::input(const char\* num)

{

HugeInteger temp(num);

\*this = temp;

}

void HugeInteger::output()

{

bool judge = 0;

for (int i = 0; i < 40; i++)

{

if (judge)

cout << integer[i];

if (integer[i] == 0 && integer[i + 1] != 0 && i + 1 < 40)

judge = 1;

else continue;

}

}

int main()

{

HugeInteger first("7654321");

HugeInteger second(7891234);

HugeInteger third(5);

first.output();

cout << " + ";

second.output();

cout << " = ";

HugeInteger res = first.add(second);

res.output();

cout << endl << endl;

second.output();

cout << " - ";

cout << 5 << " = ";

res = second.subtract(third);

res.output();

cout << endl << endl;

if(first.isEqualTo(first))

first.output();

cout << " is equal ";

first.output();

cout << endl << endl;

if (first.isNotEqualTo(second))

first.output();

cout << " is not equal to ";

second.output();

cout << endl << endl;

if(second.isGreaterThan(first))

second.output();

cout << " is greater than ";

first.output();

cout << endl << endl;

if (third.isLessThan(second))

third.output();

cout << " is less than ";

second.output();

cout << endl << endl;

if (third.isLessThanOrEqualTo(third))

third.output();

cout << " is Less Than Or Equal To ";

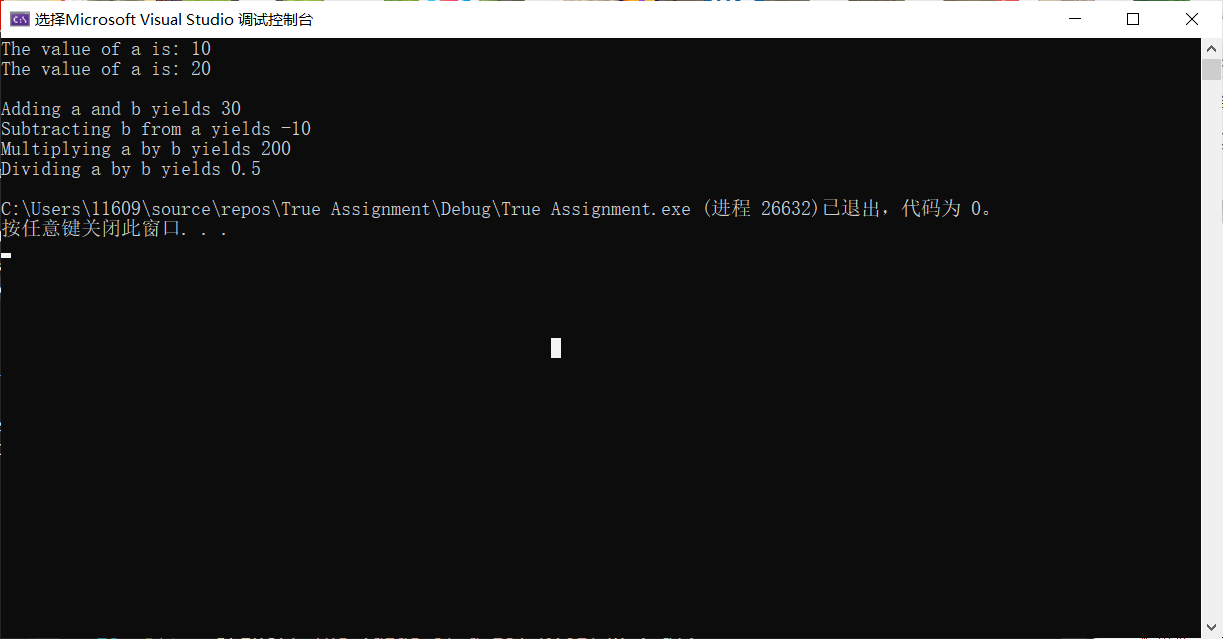
third.output();

cout << endl << endl;

}

思考与发现：转过一圈其实char数组的实现显得并不让人愉快，若是一位位进行操作，仅限制于0-9的范围，int数组方便更多，若追求动态操作，可以使用vector<int>，或采用String实现，寻求难度可以使用malloc-free/new-delete进行指针操作等，对数字大小限制也放得开，也可以方便进行改动。从钟昊原那里学到了另外两种思路，一是压位，另一是FFT（快速傅里叶变换），后者只了解了数学原理，代码实现还没多少了解。前者的尝试放在最后面。

**EX4: Simple Calculator**



#include<iostream>

#include<cstdio>

using namespace std;

class SimpleCalculator

{

public:

double add(double& left,double& right) const

{

return left + right;

}

double sub(double& left, double& right)const

{

return left - right;

}

double mul(double& left, double& right)const

{

return left \* right;

}

double div(double& left, double& right)const

{

return left / right;

}

};

int main()

{

double a = 10, b = 20;

SimpleCalculator obj;

printf("The value of a is: %.0lf\n", a);

printf("The value of a is: %.0lf\n", b);

printf("\nAdding a and b yields %.0lf\n", obj.add(a, b));

printf("Subtracting b from a yields %.0lf\n", obj.sub(a, b));

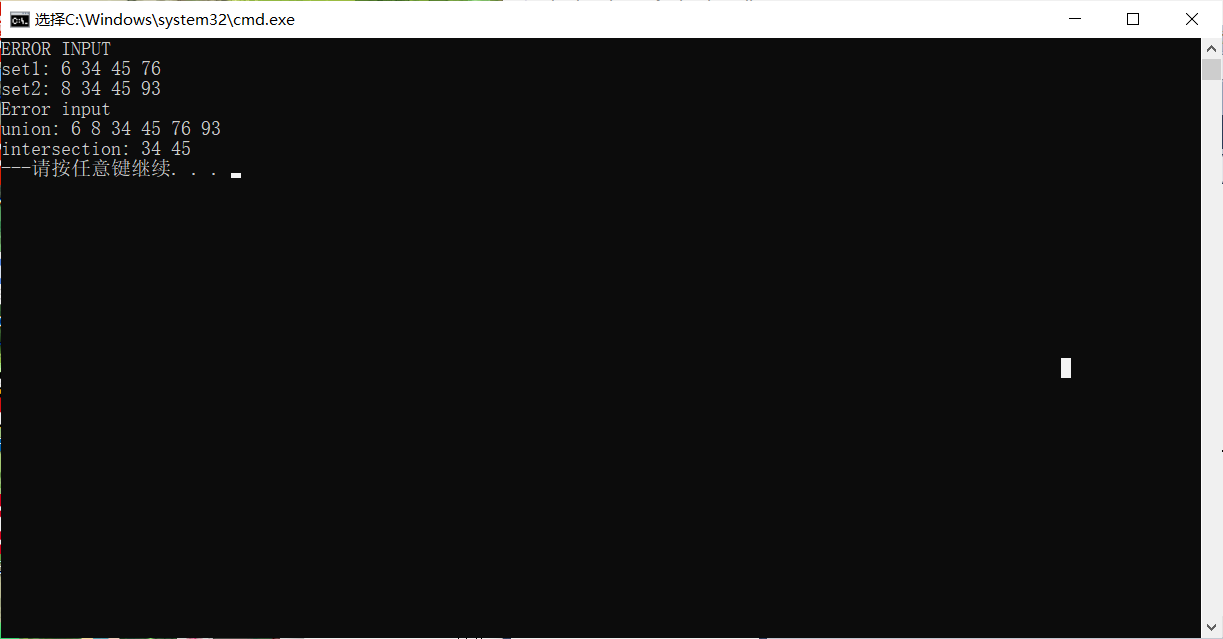
printf("Multiplying a by b yields %.0lf\n", obj.mul(a, b));

printf("Dividing a by b yields %.1lf\n", obj.div(a, b));

}

思考与发现：printf中.后接数字可以控制小数点后/字符串输出位数，格式输出很方便，也可用\*来让数字变为可变的，不过本题中double数据如果不要求精度，cout可以自行调整到有数据的那一位（不超过小数点后六位）还是更为方便，如果真有特殊要求，不妨重载<<运算符。真要写功能多的计算器不妨采用模板，重载+-\*/等运算符。

**EX5: Integer Set**



//第一次报错因为使用数组初始化，而数组中含有-1，-1未读入（非法数据），第二个为插入非法数据-1报错, 空集合输出---

#include<iostream>

using namespace std;

class IntegerSet

{

bool array[101];//此处初始化可以解决警告

bool validEntry(int m)

{

if (!this->array[m])

return 1;

else

{

printf("\nError Operation");

return 0;

}

}

public:

IntegerSet()

{

for (int i = 0; i < 101; i++)

array[i] = 0;

}

IntegerSet(int\* a, int size)

{

for (int i = 0; i < 101; i++)

array[i] = 0;

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

{

if (\*a < 0 || \*a > 100)

{

printf("ERROR INPUT");

break;

}

else this->array[\*a] = 1;

}

}

IntegerSet unionOfSets(const IntegerSet& set2) const

{

IntegerSet set3;

for (int i = 0; i < 101; i++)

{

if (this->array[i] || set2.array[i])

{

set3.array[i] = 1;

}

}

return set3;

}

IntegerSet intersectionOfSets(const IntegerSet& set2) const

{

//static 加上 & 返回,可以继续操作返回的对象

IntegerSet set3;

for (int i = 0; i < 101; i++)

{

if (this->array[i] && set2.array[i])

set3.array[i] = 1;

}

return set3;

}

void insertElement(int k)

{

if (k < 0 || k > 100)

{

cout << "Error input";

return;

}

this->array[k] = 1;

}

void deleteElement(int m)

{

if (validEntry(m))

this->array[m] = 0;

else return;

}

void printSet() const

{

bool isEmpty = 1;

for (int i = 0; i < 101; i++)

{

if (this->array[i] == 1)

{

cout << i << " ";

isEmpty = 0;

}

}

if (isEmpty)

cout << "---";

}

bool isEqualTo(const IntegerSet& set2) const

{

for (int i = 0; i < 101; i++)

if (this->array[i] != set2.array[i]) return 0;

return 1;

}

};

int main()

{

int array[20] = { 45,76,34,6,-1 };

IntegerSet set1(array, 20), set2, set3, set4;

set2.insertElement(34);

set2.insertElement(8);

set2.insertElement(93);

set2.insertElement(45);

cout << endl << "set1: ";

set1.printSet();

cout << endl;

cout << "set2: ";

set2.printSet();

cout << endl;

set2.insertElement(-1);

cout << endl;

cout << "union: ";

set3 = set1.unionOfSets(set2);

set3.printSet();

cout << endl << "intersection: ";

set3 = set1.intersectionOfSets(set2);

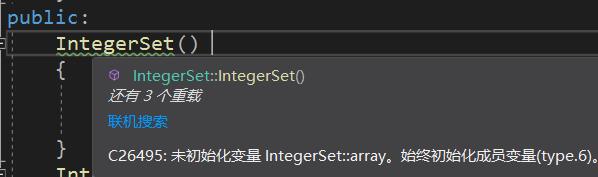
set3.printSet();

cout << endl;

set4.printSet();

}

思考与发现：在vs2010里面并没有找到c++ 标准的设置，它并不完全支持（事实上少有支持）c++ 11标准，一些用法并不能过编译，尽管一些特性并不是必须的，但确实略感不便。vs2019中如果不便在构造函数给参数默认值，而是在其内部进行初始化，下方警报是常见的（如数组）。



初始化列表

以及这样直接初始化

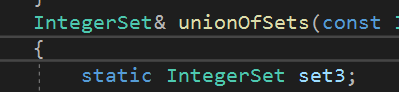


都可以解决警告。

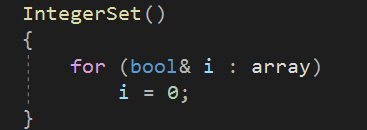
也可以委托构造实现实现多项功能（这里并没有体现，仅实现初始化后再修改数据，避免未定义）



Static加&返回，允许继续操作返回的数据（全局区存储，程序将要结束时析构），也许适用于特定情形？

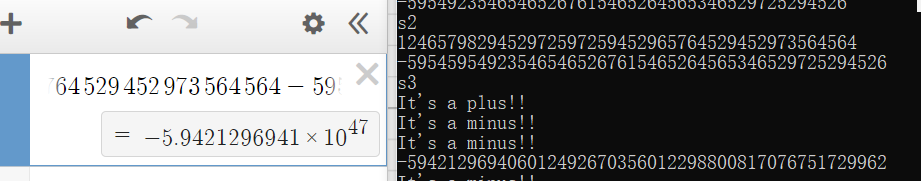


偷懒遍历，不过不能随机访问，用STL时候挺舒服

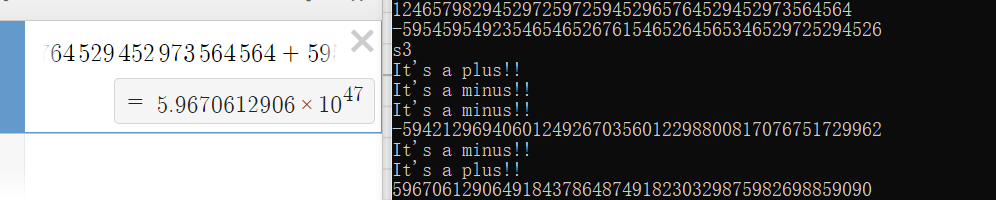


const常函数加上const传入对象，保证数据不被修改，提高效率（若不直接操作其内存地址，尝试通过这样的方式直接恶意修改，不过const对象已经被替换掉了，改了也不显示在const变量上,除非通过地址直接访问）

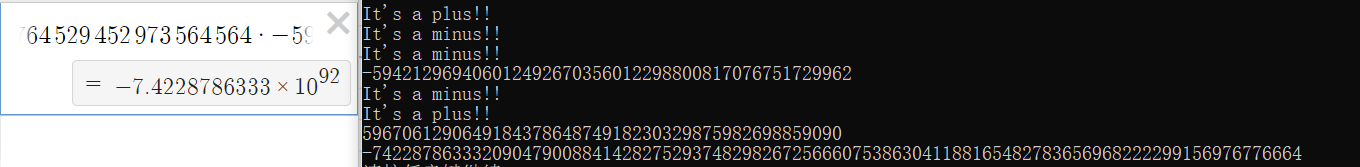
Extra：



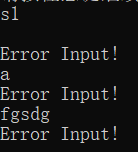
S2下两数相加



两数相减



两数相乘



合法性检查

原则上：支持负数操作，有输入的合法性检查，加减乘未做合法性检查，cin，cout，+，-，\*，>,<,==,!=进行了重载，部分用到C++11特性。

#include<iostream>

#include<vector>

#include<sstream>

#include<string>

#include<regex>

using namespace std;

constexpr const int base = 10000;

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

{

cerr << "It's a plus!!" << endl;

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 == \*this)

{

Int res2;

res2.Num.emplace\_back(0);

return res;

}

if (res > \*this)

{

res = (res - \*this);

res.isminus = 0;

return res;

}

else if (res < \*this)

{

res = (res - \*this);

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

{

cerr << "It's a minus!!" << endl;

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;

}

void input()

{

string temp;

getline(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;

}

}

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()

{

Int num1;

Int num2;

while (1)

{

cerr << "s1\n";

cin >> num1;

cin >> num2;

cerr << "s2\n";

cout << num1 << endl;

cout << num2 << endl;

cerr << "s3\n";

cout << (num1 + num2) << endl;

cout << (num1 - num2) << endl;

cout << (num1 \* num2) << endl;

system("pause");

}

}

问题与思考：代码写的过于混乱，在已经重载-（负号）的前提下，+-的条件判断完全可以更为省事。事实证明，随时测试模块调试，哪怕是完全是个人写代码，及时补上注释还是有点必要。这次写>号的时候希望用-（减法）的结果判断大小，但是-（减法）的实现又需要判断大小，一来一回就无限跳转，然后“Stack Overflow”，cerr就显示出了对-的无限调用。虽然采用压位，但实现思路上并没跟上，vector<int>显得毫无必要，4位一个整体用指针+动态分配来的更好。%与/压位下也可以方便实现，这里并未提供。