题11.9

**HugeInt Class**

题目要求：

通过学习Figs. 11.17-11.19，回答一下几个问题：

1. 描述这个程序如何实现；
2. 这个类的限制范围是多少；
3. 重载运算符 \* 和 / （乘法与除法）；
4. 重载关系运算符和等值运算符；

解题思路：

乘法运算中采取最简单的加法程序，利用for循环进行计数；除法中也采用加法程序，每次加一个除数，使得除数等于被除数时，停止操作，返回操作次数即为结果；等值运算判断依次判断每一位数字，当有一位不同时即跳出循环，若始终没有跳出循环即为相等；

源代码;

//11.9.cpp：

#include<iostream>

#include"HugeInt.h"

using namespace std;

int main()

{

HugeInt n1(7654321);

HugeInt n2(7891234);

HugeInt n3("99999999999999999999999999999");

HugeInt n4("1");

HugeInt n5;

HugeInt n6 = n1 \* n2;

cout << "n1 is " << n1 << "\nn2 is " << n2

<< "\nn3 is " << n3 << "\nn4 is " << n4

<< "\nn5 is " << n5 << "\n\n";

n5 = n1 + n2;

cout << n1 << " + " << n2 << " = " << n5 << "\n\n";

cout << n3 << " + " << n4 << "\n= " << (n3 + n4) << "\n\n";

n5 = n1 + 9;

cout << n1 << " + " << 9 << " = " << n5 << "\n\n";

n5 = n2 + "10000";

cout << n2 << " + " << "10000" << " = " << n5 << endl;

cout << "\n\nn1 \* n2 =" << n1 \* n2;

cout << "\n\nn6 / n1 =" << n6 / n1;

cout << "\n\nn1 is great than n2 ?" << (n1 > n2);

system("pause");

return 0;

}

//HugeInt.h

#ifndef HUGEINT\_H

#define HUGEINT\_H

#include<iostream>

#include<string>

using namespace std;

class HugeInt

{

friend ostream &operator<<(ostream&, const HugeInt &);

public:

static const int digits = 30;

HugeInt(long = 0);

HugeInt(const string &);

HugeInt operator+(const HugeInt&)const;

HugeInt operator+(int)const;

HugeInt operator+(const string&)const;

HugeInt operator\*(const HugeInt&) const;

HugeInt operator\*(int ) const;

HugeInt operator\*(const string&) const;

HugeInt operator/(const HugeInt&) const;

HugeInt operator/(int) const;

HugeInt operator/(const string&) const;

bool operator==(const HugeInt&) const;

bool operator>(const HugeInt&) const;

bool operator<(const HugeInt&) const;

private:

short integer[digits];

};

#endif

//HugeInt.cpp

#include "HugeInt.h"

#include<cctype>

using namespace std;

HugeInt::HugeInt(long value)

{

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

integer[i] = 0;

for (int j = digits - 1; value != 0 && j >= 0; j--)

{

integer [j] = value % 10;

value /= 10;

}

}

HugeInt::HugeInt(const string &number)

{

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

integer[i] = 0;

int length = number.size();

for (int j = digits - length, k = 0; j < digits; ++j, ++k)

if (isdigit(number[k]))

integer[j] = number[k] - '0';

}

HugeInt HugeInt::operator+(const HugeInt&op2)const

{

HugeInt temp;

int carry = 0;

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

{

temp.integer[i] = integer[i] + op2.integer[i] + carry;

if (temp.integer[i] > 9)

{

temp.integer[i] %= 10;

carry = 1;

}

else

carry = 0;

}

return temp;

}

HugeInt HugeInt::operator+(int op2)const

{

return \*this + HugeInt(op2);

}

HugeInt HugeInt::operator+(const string&op2)const

{

return \*this + HugeInt(op2);

}

ostream& operator<<(ostream &output, const HugeInt &num)

{

int i;

for (i = 0; (num.integer[i] == 0) && (i <= HugeInt::digits); ++i)

;

if (i == HugeInt::digits)

output << 0;

else

for (; i < HugeInt::digits; ++i)

output << num.integer[i];

return output;

}

HugeInt HugeInt::operator\*(const HugeInt& num) const

{

HugeInt op1;

HugeInt op2;

for (; !(op1 == num); op1 = op1 + 1)

{

op2 = op2 + \*this;

}

return op2;

}

HugeInt HugeInt::operator\*(int num) const

{

HugeInt op1(num);

HugeInt op2 = \*this \* op1;

return op2;

}

HugeInt HugeInt::operator\*(const string& str1) const

{

HugeInt op1(str1);

HugeInt op2 = \*this \* op1;

return op2;

}

HugeInt HugeInt::operator/(const HugeInt& num) const

{

HugeInt op1(0);

HugeInt op2 = num;

for (; !(op2 > \*this); op1=op1+1)

{

op2 = op2 + num;

}

return op1;

}

HugeInt HugeInt::operator/(int num) const

{

HugeInt op1(num);

HugeInt op2 = \*this / op1;

return op2;

}

HugeInt HugeInt::operator/(const string& str1) const

{

HugeInt op1(str1);

HugeInt op2 = \*this / op1;

return op2;

}

bool HugeInt::operator==(const HugeInt& num) const

{

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

{

if (integer[i] == num.integer[i])

continue;

else

return 0;

}

return 1;

}

bool HugeInt::operator>(const HugeInt& num) const

{

if (!(\*this == num))

{

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

{

if (integer[i] > num.integer[i])

return 1;

else

{

if (integer[i] == num.integer[i])

continue;

else

return 0;

}

}

}

return 0;

}

bool HugeInt::operator<(const HugeInt& num) const

{

if (!(\*this == num))

{

if (\*this > num)

return 0;

else

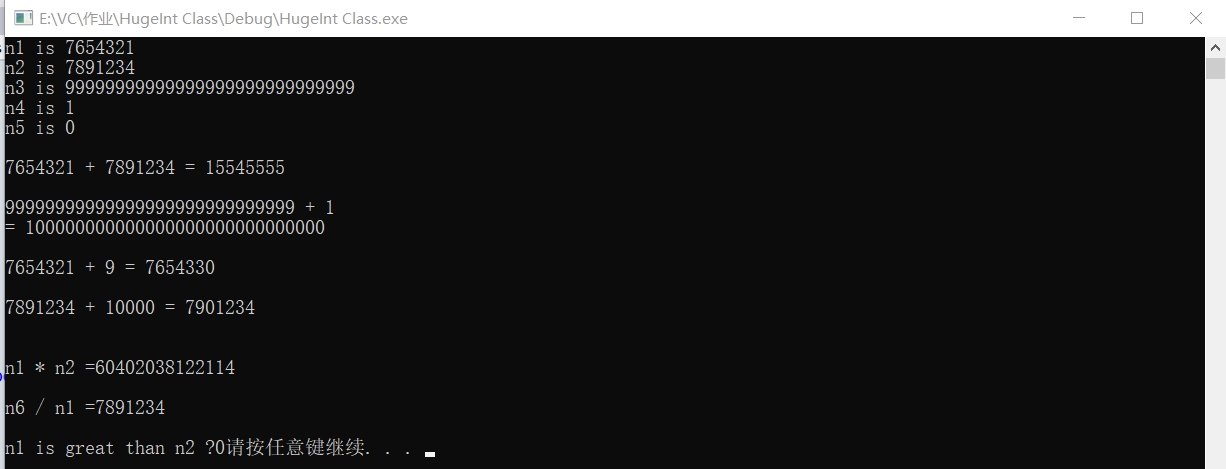
return 1;

}

return 0;

}

运行结果：



题11.10

**RationalNumber Class**

题目要求：

建立一个自然数类，要求具有一下功能：

1. 有一个构造函数防止0作为分母传入，化简不是最简的自然数，同时避免分母为负数；
2. 重载 + ，- ，\* ，/ 运算符（加减乘除法）；
3. 重载关系运算符和等值运算符；

解题思路：

设置默认构造函数，分子分母默认值均为1，在构造函数中判断，传入的分母是否为0，若为0，则输出“Change denominator from 0 to 1;”用来提醒用户，并改变分母的值为1；加减乘除运算符重载即为Rational Class中的加减乘除函数；

源代码：

//11.10.cpp

# include <iostream>

# include <string>

# include "Rational.h"

using namespace std;

int main()

{

Rational a(1, 4);

Rational b(2, 7);

cout << "a + b =" << a + b ;

cout << "a - b =" << a - b ;

cout << "a \* b ="<<a \* b;

cout << "a / b =" << a / b;

cout << "a \* 5 =" << a \* 5;

system("pause");

return 0;

}

//Rational.h

#ifndef Rational\_h

#define Rational\_h

# include <iostream>

# include <string>

using namespace std;

class Rational {

friend ostream& operator<<(ostream &, const Rational&);

int Num, Den;

public:

Rational(int num = 0, int den = 1);

Rational operator+(const Rational& ) const;

Rational operator+(int) const;

Rational operator+(const string&) const;

Rational operator-(const Rational&) const;

Rational operator-(int) const;

Rational operator-(const string&) const;

Rational operator\*(const Rational&) const;

Rational operator\*(int) const;

Rational operator\*(const string&) const;

Rational operator/(const Rational&) const;

Rational operator/(int) const;

Rational operator/(const string&) const;

bool operator==(const Rational&) const;

bool operator!=(const Rational&) const;

bool operator>(const Rational&) const;

bool operator<(const Rational&) const;

void reduce();

int StringToInt(const string&) const;

int biggest(int, int);

};

#endif

//Rational.cpp

# include <iostream>

# include <string>

# include "Rational.h"

using namespace std;

Rational::Rational(int num, int den)

{

if (den == 0)

{

den = 1;

cout << "Change denominator from 0 to 1;";

}

if (den < 0)

{

den = -den;

num = -num;

}

Num = num;

Den = den;

reduce();

}

Rational Rational::operator+(const Rational& a) const

{

Rational c;

c.Num = Num \* a.Den + Den \* a.Num;

c.Den = Den \* a.Den;

c.reduce();

return c;

}

Rational Rational::operator+(int a) const

{

Rational op1(a, 1);

Rational c = \*this + op1;

return c;

}

Rational Rational::operator+(const string& str1) const

{

int a = StringToInt(str1);

Rational c = \*this + a;

return c;

}

Rational Rational::operator-(const Rational& a) const

{

Rational c;

c.Num = Num \* a.Den - Den \* a.Num;

c.Den = Den \* a.Den;

c.reduce();

return c;

}

Rational Rational::operator-(int a) const

{

Rational op1(a, 1);

Rational c = \*this - op1;

return c;

}

Rational Rational::operator-(const string& str1) const

{

int a = StringToInt(str1);

Rational c = \*this - a;

return c;

}

Rational Rational::operator\*(const Rational& a) const

{

Rational c;

c.Num = Num \* a.Num;

c.Den = Den \* a.Den;

c.reduce();

return c;

}

Rational Rational::operator\*(int a) const

{

Rational op1(a, 1);

Rational c = \*this \*op1;

return c;

}

Rational Rational::operator\*(const string& str1) const

{

int a = StringToInt(str1);

Rational c = \*this + a;

return c;

}

Rational Rational::operator/(const Rational& a) const

{

Rational c;

c.Num = Num \* a.Den;

c.Den = Den \* a.Num;

c.reduce();

return c;

}

Rational Rational::operator/(int a) const

{

Rational op1(a, 1);

Rational c = \*this + op1;

return c;

}

Rational Rational::operator/(const string& str1) const

{

int a = StringToInt(str1);

Rational c = \*this + a;

return c;

}

bool Rational::operator==(const Rational& a) const

{

if (Num == a.Num&&Den == a.Den)

return 1;

return 0;

}

bool Rational::operator!=(const Rational& a) const

{

if (\*this == a)

return 0;

return 1;

}

bool Rational::operator>(const Rational& a) const

{

if (!(\*this == a))

{

if (Num\*a.Den > Den\*a.Num)

return 1;

else

return 0;

}

return 0;

}

bool Rational::operator<(const Rational& a) const

{

if (\*this > a)

return 0;

return 1;

}

ostream& operator<<(ostream & output, const Rational& a)

{

output << a.Num << "/" << a.Den << endl;

return output;

}

int Rational::biggest(int a, int b)

{

a = abs(a); b = abs(b);

if (a < b)

{

int temp = a;

a = b;

b = temp;

}

if (b == 0)

return a;

else

return biggest(b, a%b);

}

void Rational::reduce()

{

int gcd = biggest(Num, Den);

Num = Num / gcd;

Den = Den / gcd;

}

int Rational::StringToInt(const string& str1) const

{

int size = str1.length();

int a = 0;

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

{

a = a \* 10 + (str1[i] - '0');

}

return a;

}

运行结果：



题11.11

**Polynomial Class**

题目要求：

建立一个多项式类，用一个数组来存储多项式，每一个元素都应该包含系数和幂指数。建立一个合适的构造函数和析构函数以及set和get函数，这个类要求有以下的运算符重载：+，-，\*，/，+=，-=，\*=。

解题思路：

采用一个指针term来存储多项式的每一项，每一个下标代表幂指数，每一个数组元素表示系数；在构造函数中要求接收一个int型的数，用来表示大小，利用这个整数来创建动态数组；在乘法重载中要特别注意这个整型数的值为两个乘数的数组的大小相加，才能存储最大的项；

源代码：

//11.11.cpp

# include <iostream>

# include "Polynomial.h"

using namespace std;

int main()

{

int num1[5] = { 1,2,3,4,5 };

int num2[5] = { 5,4,3,2,1 };

Polynomial a(num1,5);

Polynomial b(num2,5);

cout << "a =" << a;

cout << "b =" << b;

Polynomial c = a + b;

Polynomial d = a - b;

Polynomial e = a \* b;

cout <<"a + b ="<< c;

cout <<"a - b =" << d;

cout << "a \* b =" << e;

system("pause");

return 0;

}

//Polynomial.h

# include <iostream>

# include <string>

# ifndef Polynomial\_H

# define Polynomial\_H

using namespace std;

class Polynomial {

friend ostream& operator<<(ostream&, const Polynomial&);

public:

Polynomial(int = 0,int =0);

Polynomial(const string& ,int size);

Polynomial(const Polynomial& );

Polynomial(int \*, int);

const Polynomial& operator=(const Polynomial&);

Polynomial operator+(const Polynomial&) const ;

Polynomial operator-(const Polynomial&) const;

Polynomial operator\*(const Polynomial&) const;

Polynomial operator+=(const Polynomial&) const;

Polynomial operator-=(const Polynomial&) const;

Polynomial operator\*=(const Polynomial&) const;

void setTerm(int, int);

int getTerm(int ) const;

private:

int Arraysize;

int \* term;

};

# endif

//Polynomial.cpp

# include <iostream>

# include <string>

# include <algorithm>

# include "Polynomial.h"

using namespace std;

Polynomial::Polynomial(int a,int size)

{

Arraysize = size;

term = new int[Arraysize];

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

setTerm(i, 0);

setTerm(0, a);

}

Polynomial::Polynomial(const string& str1,int size)

{

Arraysize = size;

term = new int[Arraysize];

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

setTerm(i, 0);

for (unsigned i=0;i<str1.length();i++)

setTerm(i, (str1[i] - '0'));

}

Polynomial::Polynomial(const Polynomial& a)

{

Arraysize = a.Arraysize;

term = new int[Arraysize];

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

{

setTerm(i, a.term[i]);

}

}

Polynomial::Polynomial(int \*a, int size)

{

Arraysize = size;

term = new int[Arraysize];

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

{

setTerm(i, a[i]);

}

}

void Polynomial::setTerm(int a, int b)

{

term[a] = b;

}

int Polynomial::getTerm(int a) const

{

return term[a];

}

const Polynomial& Polynomial::operator=(const Polynomial& a)

{

if (!(Arraysize == a.Arraysize))

{

delete[] term;

Arraysize = a.Arraysize;

term = new int[Arraysize];

}

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

{

term[i] = a.term[i];

}

return \*this;

}

Polynomial Polynomial::operator+(const Polynomial& a) const

{

Polynomial op1(0, max(Arraysize, a.Arraysize));

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

{

op1.setTerm(i, term[i] + a.term[i]);

}

return op1;

}

Polynomial Polynomial::operator+=(const Polynomial& a) const

{

Polynomial op1(0, max(Arraysize, a.Arraysize));

op1 = \*this + a;

return op1;

}

Polynomial Polynomial::operator-(const Polynomial& a) const

{

Polynomial op1(0, max(Arraysize, a.Arraysize));

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

{

op1.setTerm(i, term[i] - a.term[i]);

}

return op1;

}

Polynomial Polynomial::operator-=(const Polynomial& a) const

{

Polynomial op1(0, max(Arraysize, a.Arraysize));

op1 = \*this - a;

return op1;

}

Polynomial Polynomial::operator\*(const Polynomial& a) const

{

Polynomial op1(0, Arraysize+a.Arraysize-1);

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

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

{

op1.term[i + j] += term[i] \* a.term[j];

}

return op1;

}

Polynomial Polynomial::operator\*=(const Polynomial& a) const

{

Polynomial op1(0, Arraysize + a.Arraysize);

op1 = \*this \* a;

return op1;

}

ostream& operator<<(ostream& output, const Polynomial& a)

{

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

{

if(a.getTerm(i))

output << a.getTerm(i) << "X^" << i;

if (i == a.Arraysize -1)

break;

else

{

if (a.term[i + 1] < 0);

else

output << "+";

}

}

output << endl;

return output;

}

运行结果：

