# EX1: Complex Class

## Code

**Complex.h**

#ifndef COMPLEX\_H  
#define COMPLEX\_H  
  
#include<iostream>   
  
class Complex  
{  
 friend std::ostream &operator<<(std::ostream &,const Complex &);  
 friend std::istream &operator>>(std::istream &,Complex &);  
   
public:  
 Complex(double=0.0,double=0.0);  
 Complex operator+(const Complex&) const;  
 Complex operator-(const Complex&) const;  
 Complex operator\*(const Complex&) const;  
 Complex &operator=(const Complex&);  
 bool operator==(const Complex&) const;  
 bool operator!=(const Complex&) const;  
   
private:  
 double r;  
 double i;  
};  
  
#endif

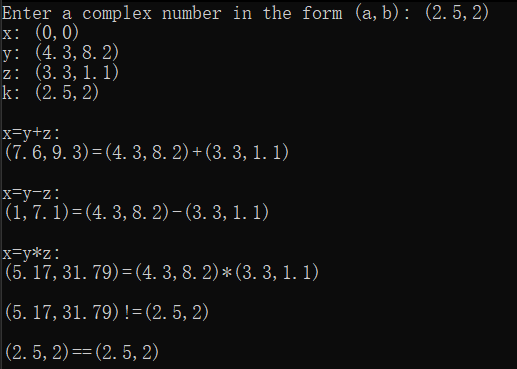
**Complex.cpp**

#include "Complex.h"  
  
std::ostream &operator<<(std::ostream &output,const Complex &c)  
{  
 output<<"("<<c.r<<","<<c.i<<")";  
 return output;  
}  
std::istream &operator>>(std::istream &input,Complex &c)  
{  
 char cc;  
 input.ignore();  
 input>>c.r;  
 input.ignore();  
 input>>c.i;  
 input.ignore();  
 return input;  
}  
   
Complex::Complex(double real,double imaginary)  
{  
 this->r=real;  
 this->i=imaginary;  
}  
  
Complex Complex::operator+(const Complex&c) const  
{  
 Complex temp(this->r + c.r,this->i + c.i);  
 return temp;  
}  
Complex Complex::operator-(const Complex&c) const  
{  
 Complex temp(this->r - c.r,this->i - c.i);  
 return temp;  
}  
Complex Complex::operator\*(const Complex&c) const  
{  
 Complex temp((this->r \* c.r)-(this->i \* c.i),  
 (this->r \* c.i)+(this->i \* c.r));  
 return temp;  
}  
  
  
Complex &Complex::operator=(const Complex&c)  
{  
 this->r = c.r;  
 this->i = c.i;  
 return \*this;  
}  
  
bool Complex::operator==(const Complex&c) const  
{  
 return ((this->r == c.r)&&(this->i == c.i));  
}  
  
bool Complex::operator!=(const Complex&c) const  
{  
 return ((this->r != c.r)||(this->i != c.i));  
}

**main.cpp**

#include "Complex.h"  
using std::cin;  
using std::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): ";  
 cin>>k;  
 cout<<"x: "<<x<<"\ny: "<<y<<"\nz: "<<z<<"\nk: "<<k<<'\n';  
 x=y+z;  
 cout<<"\nx=y+z:\n"<<x<<"="<<y<<"+"<<z<<'\n';  
 x=y-z;  
 cout<<"\nx=y-z:\n"<<x<<"="<<y<<"-"<<z<<'\n';  
 x=y\*z;  
 cout<<"\nx=y\*z:\n"<<x<<"="<<y<<"\*"<<z<<"\n\n";  
 if (x!=k)  
 cout<<x<<"!="<<k<<'\n';  
 cout<<'\n';  
 x=k;  
 if (x==k)  
 cout<<x<<"=="<<k<<'\n';  
 return 0;  
}

## Demo



# EX2: Huge Integer Class

## Code

**Hugeint.h**

#ifndef HUGEINT\_H  
#define HUGEINT\_H  
  
#include<iostream>  
  
class HugeInt  
{  
 friend std::ostream &operator<<(std::ostream &,const HugeInt &);  
// friend istream &operator>>(istream &,HugeInt &);  
public:  
 HugeInt(long long=0 );  
 HugeInt(const char \*);  
   
 HugeInt& operator=(const HugeInt &);  
   
 bool operator==(const HugeInt &) const;  
 bool operator!=(const HugeInt &) const;  
 bool operator< (const HugeInt &) const;  
 bool operator<=(const HugeInt &) const;  
 bool operator> (const HugeInt &) const;  
 bool operator>=(const HugeInt &) const;  
   
 HugeInt operator+(const HugeInt &) const;  
 HugeInt operator-(const HugeInt &) const;  
 HugeInt operator\*(const HugeInt &) const;  
 HugeInt operator/(const HugeInt &) const;  
 HugeInt operator%(const HugeInt &) const;  
   
 int getLength() const;  
  
protected:  
 int integer[40];  
};  
  
#endif

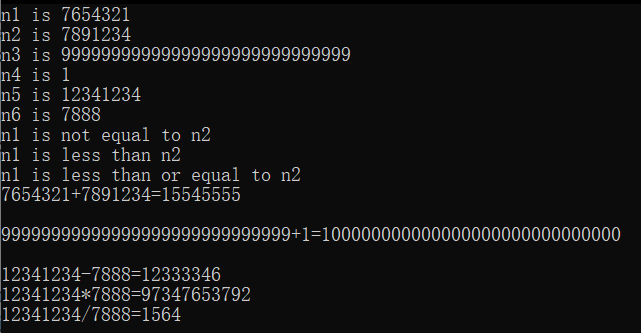
**Hugeint.cpp**

#include "Hugeint.h"  
#include<cstring>  
  
std::ostream &operator<<(std::ostream &output,const HugeInt &h)  
{  
 if (h.integer[h.integer[0]]==-1)  
 {  
 output<<"-";  
 for (size\_t i=h.integer[0]-1;i>=1;i--)  
 output<<h.integer[i];  
 }  
 else  
 for (size\_t i=h.integer[0];i>=1;i--)  
 output<<h.integer[i];  
 return output;  
}  
  
HugeInt::HugeInt(long long ilongh)  
{  
 long longh=ilongh;  
 std::memset(integer,0,sizeof(integer));  
 while (longh>0)  
 {  
 integer[0]++;  
 integer[integer[0]]=longh%10;  
 longh/=10;  
 }  
}  
HugeInt::HugeInt(const char \*h)  
{  
 memset(integer,0,sizeof(integer));  
 while (h[integer[0]]!='\0')  
 {  
 for (size\_t i=integer[0];i>=1;i--)  
 integer[i+1]=integer[i];  
 integer[0]++;  
 integer[1]=h[integer[0]-1]-'0';  
 }  
}  
   
  
HugeInt& HugeInt::operator=(const HugeInt &h)  
{  
 for (size\_t i=h.integer[0];i>=0;i--)  
 this->integer[i]=h.integer[i];  
}  
  
  
  
bool HugeInt::operator==(const HugeInt &h) const  
{  
 if (this->integer[0]!=h.integer[0])  
 return false;  
 for (size\_t i=1;i<=this->integer[0];i++)  
 if (this->integer[i]!=h.integer[i])  
 return false;  
 return true;  
}  
bool HugeInt::operator!=(const HugeInt &h) const  
{  
 if (this->integer[0]!=h.integer[0])  
 return true;  
 for (size\_t i=1;i<=this->integer[0];i++)  
 if (this->integer[i]!=h.integer[i])  
 return true;  
 return false;  
}  
   
bool HugeInt::operator< (const HugeInt &h) const  
{  
 if (this->integer[0]<h.integer[0])  
 return true;  
 if (this->integer[0]>h.integer[0])  
 return false;  
 for (size\_t i=this->integer[0];i>=1;i--)  
 {  
 if (this->integer[i]<h.integer[i])  
 return true;  
 if (this->integer[i]>h.integer[i])  
 return false;  
 }  
 return false;  
}  
bool HugeInt::operator<=(const HugeInt &h) const  
{  
 if (this->integer[0]<h.integer[0])  
 return true;  
 if (this->integer[0]>h.integer[0])  
 return false;  
 for (size\_t i=this->integer[0];i>=1;i--)  
 {  
 if (this->integer[i]<h.integer[i])  
 return true;  
 if (this->integer[i]>h.integer[i])  
 return false;  
 }  
 return true;  
}  
bool HugeInt::operator> (const HugeInt &h) const  
{  
 if (this->integer[0]>h.integer[0])  
 return true;  
 if (this->integer[0]<h.integer[0])  
 return false;  
 for (size\_t i=this->integer[0];i>=1;i--)  
 {  
 if (this->integer[i]>h.integer[i])  
 return true;  
 if (this->integer[i]<h.integer[i])  
 return false;  
 }  
 return false;  
}  
bool HugeInt::operator>=(const HugeInt &h) const  
{  
 if (this->integer[0]>h.integer[0])  
 return true;  
 if (this->integer[0]<h.integer[0])  
 return false;  
 for (size\_t i=this->integer[0];i>=1;i--)  
 {  
 if (this->integer[i]>h.integer[i])  
 return true;  
 if (this->integer[i]<h.integer[i])  
 return false;  
 }  
 return true;  
}  
  
  
HugeInt HugeInt::operator+(const HugeInt &h) const  
{  
 HugeInt temp((long long)0);  
 for (int i=1;i<=std::max(this->integer[0],h.integer[0]);i++)  
 {  
 temp.integer[i]+=(this->integer[i]+h.integer[i]);  
 temp.integer[i+1]+=(temp.integer[i]/10);  
 temp.integer[i]%=10;  
 }  
 temp.integer[0]=std::max(this->integer[0],h.integer[0])+1;  
 while (temp.integer[temp.integer[0]]==0) temp.integer[0]--;  
 return temp;  
}  
HugeInt HugeInt::operator-(const HugeInt &h) const  
{  
 HugeInt temp((long long)0);  
 for (int i=1;i<=std::max(this->integer[0],h.integer[0]);i++)  
 {  
 temp.integer[i]+=(this->integer[i]-h.integer[i]);  
 if (temp.integer[i]<0)  
 {  
 temp.integer[i+1]--;  
 temp.integer[i]+=10;  
 }  
 }  
 temp.integer[0]=std::max(this->integer[0],h.integer[0])+1;  
 while (temp.integer[temp.integer[0]]==0) temp.integer[0]--;  
 return temp;  
}  
HugeInt HugeInt::operator\*(const HugeInt &h) const  
{  
 HugeInt temp((long long)0);  
 for (int i=1;i<=this->integer[0];i++)  
 for (int j=1;j<=h.integer[0];j++)  
 temp.integer[i+j-1]+=(this->integer[i]\*h.integer[j]);  
 for (int i=1;i<=this->integer[0]+h.integer[0]+1;i++)  
 {  
 temp.integer[i+1]+=(temp.integer[i]/10);  
 temp.integer[i]%=10;  
 }  
 temp.integer[0]=this->integer[0]+h.integer[0]+1;  
 while (temp.integer[temp.integer[0]]==0) temp.integer[0]--;  
 return temp;  
}  
   
int HugeInt::getLength() const  
{  
 if (integer[integer[0]]==-1) return integer[0]-1;  
 return integer[0];  
}

**main.cpp**

#include "Hugeint.h"  
using std::cin;  
using std::cout;  
  
int main()  
  
{  
 char h1[100],h2[100],h3[100],h4[100],h5[100],h6[100];  
   
 cout<<"n1 is ";  
 cin.getline(h1,99);  
 cout<<"n2 is ";  
 cin.getline(h2,99);  
 cout<<"n3 is ";  
 cin.getline(h3,99);  
 cout<<"n4 is ";  
 cin.getline(h4,99);  
 cout<<"n5 is ";  
 cin.getline(h5,99);  
 cout<<"n6 is ";  
 cin.getline(h6,99);  
   
 HugeInt n1(h1),n2(h2),n3(h3),n4(h4),n5(h5),n6(h6);  
   
 if (n1==n2)  
 cout<<"n1 is equal to n2\n";  
 if (n1!=n2)  
 cout<<"n1 is not equal to n2\n";  
   
 if (n1<n2)  
 cout<<"n1 is less than n2\n";  
 if (n1>n2)  
 cout<<"n1 is more than n2\n";  
   
 if (n1<=n2)  
 cout<<"n1 is less than or equal to n2\n";  
 if (n1>=n2)  
 cout<<"n1 is more than or equal to n2\n";  
   
 cout<<n1<<"+"<<n2<<"="<<n1+n2<<"\n\n";  
 cout<<n3<<"+"<<n4<<"="<<n3+n4<<"\n\n";  
   
 cout<<n5<<"-"<<n6<<"="<<n5-n6<<"\n";  
 cout<<n5<<"\*"<<n6<<"="<<n5\*n6<<"\n";  
 cout<<n5<<"/"<<n6<<"="<<n5/n6<<"\n";  
}  
/\*  
7654321  
7891234  
99999999999999999999999999999  
1  
12341234  
7888  
  
\*/

## Demo



# EX3: Rational Number Class

## Code

**RationalNumber.h**

#ifndef RATIONALNUMBER\_H  
#define RATIONALNUMBER\_H  
  
  
class RationalNumber  
{  
public:   
 RationalNumber(int=0,int=1);  
   
 RationalNumber operator+(const RationalNumber&);  
 RationalNumber operator-(const RationalNumber&);  
 RationalNumber operator\*(const RationalNumber&);  
 RationalNumber operator/( RationalNumber&);  
  
 bool operator> (const RationalNumber&) const;  
 bool operator< (const RationalNumber&) const;  
 bool operator>=(const RationalNumber&) const;  
 bool operator<=(const RationalNumber&) const;  
   
 bool operator==(const RationalNumber&) const;  
 bool operator!=(const RationalNumber&) const;  
  
 void printRational() const;  
   
private:  
 int n;  
 int d;  
 void reduction();   
};  
  
#endif

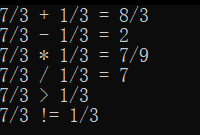
**RationalNumber.cpp**

#include "RationalNumber.h"  
#include<iostream>  
  
RationalNumber::RationalNumber(int numerator,int denominator)  
{  
 n=numerator;  
 if (denominator!=0) d=denominator;  
 else d=1;  
 this->reduction();  
}  
   
RationalNumber RationalNumber::operator+(const RationalNumber& r)  
{  
 RationalNumber temp(this->n \* r.d + this->d \* r.n,this->d \* r.d);  
 temp.reduction();  
 return temp;  
}  
RationalNumber RationalNumber::operator-(const RationalNumber& r)  
{  
 RationalNumber temp(this->n \* r.d - this->d \* r.n,this->d \* r.d);  
 temp.reduction();  
 return temp;  
}  
RationalNumber RationalNumber::operator\*(const RationalNumber& r)  
{  
 RationalNumber temp(this->n \* r.n,this->d \* r.d);  
 temp.reduction();  
 return temp;  
}  
RationalNumber RationalNumber::operator/( RationalNumber& r)  
{  
 if (r.d==0)  
 {  
 std::cout<<"WRONG!";  
 }  
 else  
 {  
 RationalNumber temp(this->n \* r.d,this->d \* r.n);  
 temp.reduction();  
 return temp;  
 }  
}  
  
bool RationalNumber::operator> (const RationalNumber& r) const  
{  
 return (this->n \* r.d - this->d \* r.n > 0);  
}  
bool RationalNumber::operator< (const RationalNumber& r) const  
{  
 return (this->n \* r.d - this->d \* r.n < 0);  
}  
bool RationalNumber::operator>=(const RationalNumber& r) const  
{  
 return (this->n \* r.d - this->d \* r.n >=0);  
}  
bool RationalNumber::operator<=(const RationalNumber& r) const  
{  
 return (this->n \* r.d - this->d \* r.n >=0);  
}  
  
bool RationalNumber::operator==(const RationalNumber& r) const  
{  
 return (this->n \* r.d == this->d \* r.n);  
}  
bool RationalNumber::operator!=(const RationalNumber& r) const  
{  
 return (this->n \* r.d == this->d \* r.n);  
}  
  
void RationalNumber::printRational() const  
{  
 if (d==1) std::cout<< this->n ;  
 else std::cout<< this->n <<'/'<< this->d ;  
}  
   
void RationalNumber::reduction()  
{  
 for (int i=2;i<=std::min(n,d);i++)  
 if (n%i==0 && d%i==0)  
 {  
 n/=i;  
 d/=i;  
 }  
}

**main.cpp**

#include "RationalNumber.h"  
#include<iostream>  
using std::cout;  
  
int main()  
{  
 RationalNumber r1(7,3);  
 RationalNumber r2(1,3);  
 RationalNumber r3;  
 r3=r1+r2;  
 r1.printRational();  
 cout<<" + ";  
 r2.printRational();  
 cout<<" = ";  
 r3.printRational();  
 cout<<'\n';  
   
 r3=r1-r2;  
 r1.printRational();  
 cout<<" - ";  
 r2.printRational();  
 cout<<" = ";  
 r3.printRational();  
 cout<<'\n';  
   
 r3=r1\*r2;  
 r1.printRational();  
 cout<<" \* ";  
 r2.printRational();  
 cout<<" = ";  
 r3.printRational();  
 cout<<'\n';  
   
 r3=r1/r2;  
 r1.printRational();  
 cout<<" / ";  
 r2.printRational();  
 cout<<" = ";  
 r3.printRational();  
 cout<<'\n';  
   
 r1.printRational();  
 cout<<((r1<r2)?" < ":" > ");  
 r2.printRational();  
 cout<<'\n';  
   
 r1.printRational();  
 cout<<((r1==r2)?" = ":" != ");  
 r2.printRational();  
 cout<<'\n';  
}

## Demo



# EX4: String Concatenation

## Code

**main.cpp**

#include <iostream>   
#include <cstdio>   
#include <cstring>   
using namespace std;   
   
class String   
{   
public:   
 friend ostream &operator<<(ostream &output,const String &s)   
 {   
 output<<s.sPtr;   
 return output;   
 }   
   
 String(char \*a=(char\*)"")   
 {   
 length=strlen(a);   
 sPtr=new char[length+1];   
 strcpy(sPtr,a);   
 }   
   
 String &operator=(String& a)   
 {   
 if(this->sPtr==a.sPtr)   
 return \*this;   
 length=a.length;   
 delete []sPtr;   
 sPtr=new char[length];   
 strcpy(sPtr,a.sPtr);   
 return \*this;   
 }   
   
 ~String()   
 {   
 delete []sPtr;   
 }   
   
 friend String& operator+(const String &a,const String &b)   
 {   
 static String r;   
 r.length=a.length+b.length;   
 r.sPtr=new char[r.length+1];   
 for (int j=0;j<a.length;j++)   
 r.sPtr[j]=a.sPtr[j];   
 for (int i=0;i<b.length;i++)   
 r.sPtr[i+a.length]=b.sPtr[i];   
 r.sPtr[r.length]='\0';   
 return r;   
 }   
   
private:   
 char \*sPtr;   
 int length;   
};   
   
int main()   
{   
 String string1,string2((char\*)"The date is");   
 String string3((char\*)" August 1, 1993");   
 cout<<"string1=string2 + string3\n";   
 string1=string2+string3;   
 cout<<"\""<<string1<<"\"=\""<<string2<<"\" + \""<<string3<<"\""<<endl;   
 return 0;   
}

## Demo

image-20210405062324677