# EX1: Complex Class

## Code

**Complex.h**

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

**Complex.cpp**

#include "Complex.h"   
   
ostream &operator<<(ostream &output,const Complex &c)   
{   
 output<<"("<<c.r<<","<<c.i<<")";   
 return output;   
}   
   
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;   
}

**main.cpp**

#include "Complex.h"   
using std::cin;   
using std::cout;   
   
int main()   
{   
 Complex x(1,7),y(9,2),z(10,1),t(11,5);   
 cout<<x<<'+'<<y<<'='<<x+y<<'\n';   
 cout<<z<<'-'<<t<<'='<<z-t<<'\n';   
 return 0;   
}

## Demo



# EX2: Enhancing Time Class

## Code

**Time.h**

#ifndef TIME\_H   
#define TIME\_H   
   
class Time   
{   
public:   
 friend void tick(Time &t);   
   
 explicit Time(int=0,int=0,int=0);   
   
 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

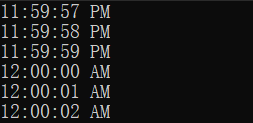
**Time.cpp**

#include<iostream>   
#include<iomanip>   
#include<stdexcept>   
#include<windows.h>   
#include "Time.h"   
   
using std::invalid\_argument;   
   
void tick(Time &t)   
{   
 while (true)   
 {   
 t.printStandard();   
 if (t.getSecond()==59)   
 {   
 t.setSecond(0);   
 if (t.getMinute()==59)   
 {   
 t.setMinute(0);   
 if (t.getHour()==23)   
 t.setHour(0);   
 else   
 t.setHour(t.getHour()+1);   
 }   
 else   
 t.setMinute(t.getMinute()+1);   
 }   
 else   
 t.setSecond(t.getSecond()+1);   
   
 Sleep(1000);   
 }   
}   
   
Time::Time(int hour,int minute,int second)   
{   
 setTime(hour,minute,second);   
}   
   
void Time::setTime(int h,int m,int s)   
{   
 if ((h>=0 && h<24) && (m>=0 && m<60) && (s>=0 && s<60))   
 {   
 hour=h;   
 minute=m;   
 second=s;   
 }   
 else   
 throw invalid\_argument("hour,minute and/or second was out of range");   
}   
   
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   
{   
 std::cout<<std::setfill('0')<<std::setw(2)<<hour<<":"   
 <<std::setw(2)<<minute<<":"<<std::setw(2)<<second;   
}   
   
void Time::printStandard() const   
{   
 std::cout<<((hour==0 || hour==12)?12:hour%12)<<":"   
 <<std::setfill('0')<<std::setw(2)<<minute<<":"   
 <<std::setw(2)<<second<<(hour<12?" AM":" PM")<<std::endl;   
}

**main.cpp**

#include<iostream>   
#include<stdexcept>   
#include "Time.h"   
   
int main()   
{   
 Time t(23,59,57);   
 tick(t);   
}

## Demo



# EX3: Huge Integer Class

## Code

**HugeInteger.h**

#ifndef HUGEINTEGER\_H   
#define HUGEINTEGER\_H   
   
#include<iostream>   
   
class HugeInteger   
{   
public:   
 void output();   
   
 HugeInteger(int=0);   
 HugeInteger(const char \*);   
   
 bool isEqualTo(const HugeInteger &);   
 bool isNotEqualTo(const HugeInteger &);   
 bool isLessThan(const HugeInteger &);   
 bool isLessThanOrEqualTo(const HugeInteger &);   
 bool isGreaterThan(const HugeInteger &);   
 bool isGreaterThanOrEqualTo(const HugeInteger &);   
   
 HugeInteger add(const HugeInteger &);   
 HugeInteger add(int);   
 HugeInteger add(const char \*);   
   
 HugeInteger substract(const HugeInteger &);   
 HugeInteger substract(int);   
 HugeInteger substract(const char \*);   
   
 bool isZero();   
   
 int getLength();   
   
protected:   
 int integer[40];   
};   
   
#endif

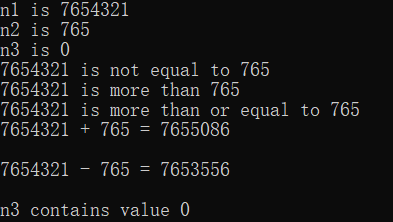
**HugeInteger.cpp**

#include "HugeInteger.h"   
   
#include<iostream>   
#include<cstring>   
#include<algorithm>   
   
void HugeInteger::output()   
{   
 for (size\_t i=this->integer[0];i>=1;i--)   
 std::cout<<this->integer[i];   
}   
   
   
   
HugeInteger::HugeInteger(int ilongh)   
{   
 long longh=ilongh;   
 memset(integer,0,sizeof(integer));   
 while (longh>0)   
 {   
 integer[0]++;   
 integer[integer[0]]=longh%10;   
 longh/=10;   
 }   
}   
HugeInteger::HugeInteger(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';   
 }   
}   
   
   
   
bool HugeInteger::isEqualTo(const HugeInteger &h)   
{   
 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 HugeInteger::isNotEqualTo(const HugeInteger &h)   
{   
 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 HugeInteger::isLessThan(const HugeInteger &h)   
{   
 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 HugeInteger::isLessThanOrEqualTo(const HugeInteger &h)   
{   
 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 HugeInteger::isGreaterThan(const HugeInteger &h)   
{   
 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 HugeInteger::isGreaterThanOrEqualTo(const HugeInteger &h)   
{   
 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;   
}   
   
   
HugeInteger HugeInteger::add(const HugeInteger &h)   
{   
 HugeInteger temp((int)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;   
}   
   
HugeInteger HugeInteger::add(int i)   
{   
 HugeInteger temp((int)i);   
 return this->add(temp);   
}   
   
HugeInteger HugeInteger::add(const char \*c)   
{   
 HugeInteger temp(c);   
 return this->add(temp);   
}   
   
   
   
HugeInteger HugeInteger::substract(const HugeInteger &h)   
{   
 HugeInteger temp((int)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;   
}   
   
HugeInteger HugeInteger::substract(int i)   
{   
 HugeInteger temp((int)i);   
 return this->substract(temp);   
}   
   
HugeInteger HugeInteger::substract(const char \*c)   
{   
 HugeInteger temp(c);   
 return this->substract(temp);   
}   
   
   
bool HugeInteger::isZero()   
{   
 return (this->integer[0])==1;   
}   
   
   
   
int HugeInteger::getLength()   
{   
 if (integer[integer[0]]==-1) return integer[0]-1;   
 return integer[0];   
}

**main.cpp**

#include "HugeInteger.h"   
   
#include<iostream>   
using std::cin;   
using std::cout;   
   
int main()   
{   
 char h1[100],h2[100],h3[100];   
   
 cout<<"n1 is ";   
 cin.getline(h1,99);   
 cout<<"n2 is ";   
 cin.getline(h2,99);   
 cout<<"n3 is ";   
 cin.getline(h3,99);   
   
 HugeInteger n1(h1),n2(h2),n3(h3);   
   
 if (n1.isEqualTo(n2))   
 n1.output(),std::cout<<" is equal to ",n2.output(),std::cout<<std::endl;   
 if (n1.isNotEqualTo(n2))   
 n1.output(),std::cout<<" is not equal to ",n2.output(),std::cout<<std::endl;   
   
 if (n1.isLessThan(n2))   
 n1.output(),std::cout<<" is less than ",n2.output(),std::cout<<std::endl;   
 if (n1.isGreaterThan(n2))   
 n1.output(),std::cout<<" is more than ",n2.output(),std::cout<<std::endl;   
   
 if (n1.isLessThanOrEqualTo(n2))   
 n1.output(),std::cout<<" is less than or equal to ",n2.output(),std::cout<<std::endl;   
 if (n1.isGreaterThanOrEqualTo(n2))   
 n1.output(),std::cout<<" is more than or equal to ",n2.output(),std::cout<<std::endl;   
   
 n1.output();   
 std::cout<<" + ";   
 n2.output();   
 std::cout<<" = ";   
 (n1.add(n2)).output();   
 std::cout<<"\n\n";   
   
 n1.output();   
 std::cout<<" - ";   
 n2.output();   
 std::cout<<" = ";   
 (n1.substract(n2)).output();   
 std::cout<<"\n\n";   
   
 if (n3.isZero())   
 std::cout<<"n3 contains value 0";   
}

## Demo



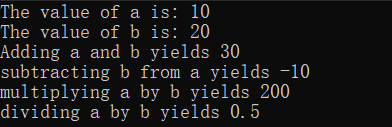
# EX4: Simple Calculator

## Code

**main.cpp**

#include<iostream>   
   
using std::cout;   
using std::endl;   
   
class SimpleCalculator   
{   
public:   
 double add(double a,double b)   
 {   
 return a+b;   
 }   
 double sub(double a,double b)   
 {   
 return a-b;   
 }   
 double mul(double a,double b)   
 {   
 return a\*b;   
 }   
 double div(double a,double b)   
 {   
 return a/b;   
 }   
};   
   
int main()   
{   
 double a=10;   
 double b=20;   
 SimpleCalculator sc;   
 cout<<"The value of a is: " <<a<<endl;   
 cout<<"The value of b is: " <<b<<endl;   
   
 cout<<"Adding a and b yields "<<sc.add(a,b)<<endl;   
 cout<<"subtracting b from a yields "<<sc.sub(a,b)<<endl;   
 cout<<"multiplying a by b yields "<<sc.mul(a,b)<<endl;   
 cout<<"dividing a by b yields "<<sc.div(a,b)<<endl;   
}

## Demo



# EX5: Integer Set

## Code

**IntegerSet.h**

#ifndef INTEGERSET\_H   
#define INTEGERSET\_H   
   
#include<cstring>   
   
class IntegerSet   
{   
private:   
 int k[101];   
public:   
 IntegerSet()   
 {   
 memset(k,0,sizeof(k));   
 }   
   
 IntegerSet(int\*,int);   
   
 void inputSet();   
 void printSet() const;   
 bool isEqualTo(IntegerSet) const;   
 void insertElement(int);   
 IntegerSet unionOfSets(IntegerSet) const;   
 IntegerSet intersectionOfSets(IntegerSet) const;   
 void deleteElement(int);   
   
 void printEqual(bool);   
};   
   
#endif

**IntegerSet.cpp**

#include "IntegerSet.h"   
   
#include<iostream>   
using std::cin;   
using std::cout;   
using std::endl;   
   
IntegerSet::IntegerSet(int\* num,int n)   
{   
 for (int i=0;i<n;++i)   
 k[num[i]]=1;   
}   
   
void IntegerSet::inputSet()   
{   
 int inp=0;   
 cout<<"Enter an element ('-1' to end) :";   
 cin>>inp;   
 while (inp!=-1)   
 {   
 k[inp]=1;   
 cout<<"Enter an element ('-1' to end) :";   
 cin>>inp;   
 }   
 cout<<"Entry complete\n";   
 cout<<endl;   
}   
   
void IntegerSet::printSet() const   
{   
 cout<<"{ ";   
 for (int i=0;i<101;++i)   
 if(k[i]==1)   
 cout<<i<<" ";   
 cout<<"}"<<endl;   
}   
   
bool IntegerSet::isEqualTo(IntegerSet a)const   
{   
 for (int i=0;i<101;++i)   
 if(k[i]!=a.k[i])   
 return false;   
 return true;   
}   
   
void IntegerSet::insertElement(int n)   
{   
 k[n]=1;   
}   
   
IntegerSet IntegerSet::unionOfSets(IntegerSet a) const   
{   
 IntegerSet added=a;   
 for (int i=0;i<101;++i)   
 if(k[i]==1)   
 added.k[i]=1;   
 return added;   
}   
   
IntegerSet IntegerSet::intersectionOfSets(IntegerSet a) const   
{   
 IntegerSet iso;   
 for (int i=0;i<101;++i)   
 if(a.k[i]==1 && this->k[i]==1)   
 iso.k[i]=1;   
 return iso;   
}   
   
void IntegerSet::deleteElement(int n)   
{   
 if(k[n]==1)   
 k[n]=0;   
 else   
 cout<<"Invalid insert attempted!"<<endl;   
}   
   
void IntegerSet::printEqual(bool a)   
{   
 if(a==1)   
 cout<<"A is equal to B\n";   
 else   
 cout<<"A is not equal to B\n";   
}

**main.cpp**

#include "IntegerSet.h"   
   
#include<iostream>   
using std::cout;   
using std::endl;   
const int arraySize=10;   
   
int main()   
{   
 IntegerSet A;   
 IntegerSet B;   
 IntegerSet C;   
   
 cout<<"Enter set A:\n";   
 A.inputSet();   
 cout<<"Enter set B:\n";   
 B.inputSet();   
   
 cout<<"Union of A and B is:\n";   
 C=A.unionOfSets(B);   
 C.printSet();   
   
 cout<<"intersection of A and B is:\n";   
 C=A.intersectionOfSets(B);   
 C.printSet();   
   
 A.printEqual(A.isEqualTo(B));   
 cout<<endl;   
   
 cout<<"insert 77 into set A...\n";   
 A.insertElement(77);   
 cout<<"Set A is now :\n";   
 A.printSet();   
 cout<<endl;   
   
 cout<<"delete 77 from A...\n";   
 A.deleteElement(77);   
 cout<<"Set A is now :\n";   
 A.printSet();   
 A.deleteElement(101);   
 A.deleteElement(50);   
   
 const int arraySize=10;   
 int intArray[arraySize]={25,67,2,9,99,105,45,-5,101,1};   
 IntegerSet e(intArray,arraySize);   
   
 cout<<"\nSet e is:\n";   
 e.printSet();   
}

## Demo

