实验五 运算符重载与类模板

实验目的

1. 掌握运算符重载的概念；
2. 掌握使用 friend 重载运算符的方法；
3. 掌握使用成员函数重载运算符的方法；
4. 掌握类模板的使用方法；

实验内容

1. Complex 类的设计

要求：

1）重载+、－、* 和 /（使用成员函数和全局函数两种方式），以支持复数的算术运算

2）重载<<和>>运算符，从键盘输入一个复数，并可以输出一个复数至标准输出

3）设计默认构造函数，将实部和虚 部设为零

4）设计拥有一个参数的构造函数，将实部设为该参数，虚部设为零

5）设计拥有两个参数的构造函数，将两个参数分别赋给实部和虚部

6）重载前置与后置的++、－－运算符

7）重载一个转型运算符

[参考代码]
Complex.h

```
#pragma once
```

```
#include <iostream>
using namespace std;
class Complex
{
public:
  Complex(double r=0, double i=0)
    :real(r),imag(i) {}
  virtual ~Complex(void);
//operator double()  {return real;}
   // operator methods
//   Complex operator+( const
Complex& ) const;
   Complex operator-( const Complex& )
const;
   Complex operator*( const Complex& )
const;
   Complex operator/( const Complex& )
const;
   friend istream&  operator >>
( istream& in,  Complex& c);
   friend ostream&  operator <<
( ostream& out,  Complex& c);
```

```
  double get_real() const { return
real;}
  double get_imag() const { return
imag;}
   Complex operator++( );
   Complex operator++( int );
private:
   double real;
   double imag;
};
istream&  operator >> ( istream& in,
Complex& c);
ostream&  operator << ( ostream& out,
Complex& c);
Complex operator + (const Complex& t,
const Complex& u);
```

[Complex.cpp]

```
#include "Complex.h"



Complex::~Complex(void)
```

```
{
}

// Complex + as binary operator
//Complex Complex::operator+( const
Complex& u ) const {
//   Complex v( real + u.real,
//              imag + u.imag );
//   return v;
//}

Complex operator + (const Complex& t,
const Complex& u)
{
  return Complex( t.get_real() +
u.get_real(),
                          t.get_imag()
+ u.get_imag() );
}

// Complex - as binary operator
Complex Complex::operator-( const
```

```
Complex& u ) const {
   Complex v( real - u.real,
              imag - u.imag );
   return v;
}

// Complex * as binary operator
Complex Complex::operator*( const
Complex& u ) const {
   Complex v( real * u.real - imag *
u.imag,
              imag * u.real + real *
u.imag );
   return v;
}

// Complex / as binary operator
Complex Complex::operator/( const
Complex& u ) const {
   double abs_sq = u.real * u.real +
u.imag * u.imag;
   Complex v( ( real * u.real + imag *
```

```
u.imag ) / abs_sq,
                ( imag * u.real - real *
u.imag ) / abs_sq );
   return v;
}

istream&  operator >> ( istream& in,
Complex& c) {
  return      in >> c.real>>c.imag;
}

ostream& operator << ( ostream& out,
Complex& c) {
  return      out<< c.real
<<"+"<<c.imag<<"i";
}
Complex Complex::operator++( )//前置
{
  real += 1;
  imag += 1;
  return *this;
}
```

```
Complex Complex::operator++( int )//后置
{
  Complex  temp(*this);
  real += 1;
  imag += 1;
  return temp;
}
```

2. Array<T>和 TwoArray<T>类模板的设计

要求：

1）为一维数组 Array 重载下标[]运算符，为二维数组 TwoArray 重载函数 () 运算符以支持对数组元素的存取

2）重载<<和>>运算符，从键盘输入一维数组和二维数组，并可以输出一维数组和二维数组元素至标准输出

3）设计一维数组和二维数组的拷贝构造函数

4）重载一维数组和二维数组的赋值=运算符

[参考代码]
intArray.h

```
#pragma once

class intArray
{
```

```
public:
  intArray(void) : size(0), a(0) {};
  intArray(int s);
  intArray(const intArray&);
  virtual ~intArray(void);
  int& operator[]( int );
  const int& operator[]( int ) const;
  intArray& operator= (const
intArray&);
  int get_size() const { return size;}
private:
  int size;
  int* a;
  void copyIntoP(const intArray&);
};
```

intArray.cpp

```
#include "intArray.h"
#include <string>
using namespace std;
intArray::intArray(int s)
{
  a = new int[ s ];
```

```
  size = s;
}

intArray::~intArray(void)
{
  delete [] a;
}

int&   intArray::operator [] ( int i )
{
  if (i < 0 || i >= size)
    throw string( "OutOfBounds" );
  return a[ i ];
}

const int& intArray::operator [] ( int
i ) const
{
  if (i < 0 || i >= size)
    throw string( "OutOfBounds" );
  return a[ i ];
}
```

```
intArray::intArray(const intArray& d)
:size(0),a(0)
{
  copyIntoP(d);
}

intArray& intArray::operator= (const
intArray& d)
{
  if (this != &d)
    copyIntoP(d);
  return *this;
}

void  intArray::copyIntoP(const
intArray& d) {
  if(a!=NULL)  delete [] a;
  if (d.a != NULL) {
    a = new int[size = d.size];
    for (int i=0; i<size; i++)
      a[i] = d[i];//d.a[i];
```

```
    }
    else{
      a = NULL;
      size = 0;
    }
  }
```

intTwoArray.h

```
#pragma once

class intTwoArray
{
public:
  int&  operator()  (int,  int);
  const  int& operator() (int, int)
const;
  intTwoArray( int s1,  int s2);
  virtual ~intTwoArray(void);
  int  get_size1()  const  {  return
size1; }
  int  get_size2()  const  {  return
size2; }
private:
```

```
    int size1;
    int size2;
    int*  a;
};
```

intTwoArray.cpp

```
#include "intTwoArray.h"
#include <string>
using namespace std;
intTwoArray::intTwoArray( int s1,  int
s2)
{
    int  size = s1 * s2;
    a = new int  [size];
    size1 = s1;
    size2 = s2;
}


intTwoArray::~intTwoArray(void)
{
```

```
  delete [] a;
}

int&  intTwoArray::operator ()  (int i,
int j)
{
  if (i<0 || i >= size1)
    throw string( "FirstOutOfBounds");
  if  (j<0 || j >= size2)
    throw string("SecondOutOfBounds");
  return a[ i * size2 + j];
}

const int&  intTwoArray::operator()
(int i, int j) const
{
  if (i<0 || i >= size1)
    throw string("FirstOutOfBounds");
  if  (j<0 || j >= size2)
    throw string("SecondOutOfBounds");
  return a[ i * size2 + j];
}
```

Test.cpp

```
#include <iostream>
#include "Complex.h"
#include "Array.h"
#include "TwoArray.h"
using namespace std;

int main()
{
    std::cout << "Hello Template
World!\n";
  Array<Complex> ca(10);
  TwoArray<Complex> cta(10, 5);

  int i,j;
  for (i = 0; i < 10; i++)
  {
    ca[i].set_real(10.0*i);
    ca[i].set_imag(i / 10.0);
    cout << ca[i]<<endl;
  }
  for (i = 0; i < 10; i++)
```

```
    {
        for (j = 0; j < 5; j++)
        {
            cta(i,j).set_real(10.0 * i);
            cta(i,j).set_imag(j / 10.0);
            cout << cta(i, j) << '\t';
        }
        cout << endl;
    }
}
```