实验八：运算符重载&继承

实验目的

1. 掌握运算符重载的定义、限制和使用
2. 掌握派生类的定义，对基类的使用和限制

实验作业

作业一（Complex 类）

1. 问题描述

(Complex Class) Consider class **Complex** that enables operations on so-called complex numbers. These are numbers of the form realPart + imaginaryPart * *i*, where *i* has the value $\sqrt{-1}$

a) Modify the class to enable input and output of complex numbers via overloaded >> and << operators, respectively (you should remove the print function from the class).
b) Overload the multiplication operator to enable multiplication of two complex numbers as in algebra.
c) Overload the == and != operators to allow comparisons of complex numbers.

2. 实验提示

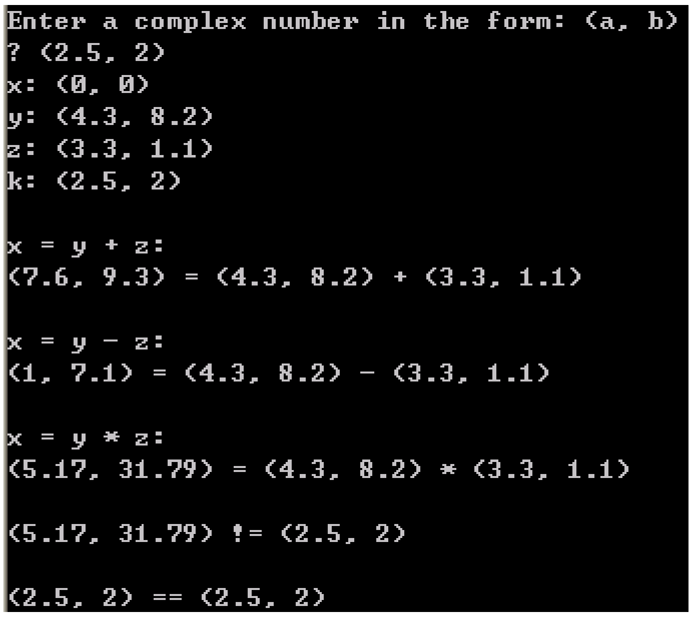
1）类定义示例：

```
#ifndef COMPLEX_H
#define COMPLEX_H

#include <iostream>
using std::ostream;
using std::istream;

class Complex
{
    friend ostream &operator<<( ostream &, const Complex & );
    friend istream &operator>>( istream &, Complex & );
public:
    Complex( double = 0.0, double = 0.0 ); // constructor
    Complex operator+( const Complex& ) const; // addition
    Complex operator-( const Complex& ) const; // subtraction
    Complex operator*( const Complex& ) const; // multiplication
    Complex& operator=( const Complex& ); // assignment
    bool operator==( const Complex& ) const;
    bool operator!=( const Complex& ) const;
private:
    double real; // real part
    double imaginary; // imaginary part
}; // end class Complex

#endif
```

2）测试函数示例

```
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)\n? ";
    cin >> k; // demonstrating overloaded >>
    cout << "x: " << x << "\ny: " << y << "\nz: " << z << "\nk: "
        << k << '\n'; // demonstrating overloaded <<
    x = y + z; // demonstrating overloaded + and =
    cout << "\nx = y + z:\n" << x << " = " << y << " + " << z << '\n';
    x = y - z; // demonstrating overloaded - and =
    cout << "\nx = y - z:\n" << x << " = " << y << " - " << z << '\n';
    x = y * z; // demonstrating overloaded * and =
    cout << "\nx = y * z:\n" << x << " = " << y << " * " << z << "\n\n";

    if ( x != k ) // demonstrating overloaded !=
        cout << x << " != " << k << '\n';

    cout << '\n';
    x = k;

    if ( x == k ) // demonstrating overloaded ==
        cout << x << " == " << k << '\n';

    return 0;
} // end main
```

3. 结果示例



```
Enter a complex number in the form: (a, b)
? (2.5, 2)
x: (0, 0)
y: (4.3, 8.2)
z: (3.3, 1.1)
k: (2.5, 2)

x = y + z:
(7.6, 9.3) = (4.3, 8.2) + (3.3, 1.1)

x = y - z:
(1, 7.1) = (4.3, 8.2) - (3.3, 1.1)

x = y * z:
(5.17, 31.79) = (4.3, 8.2) * (3.3, 1.1)

(5.17, 31.79) != (2.5, 2)

(2.5, 2) == (2.5, 2)
```

作业二（习题 10.9，Hugeint 类）

1. 问题描述

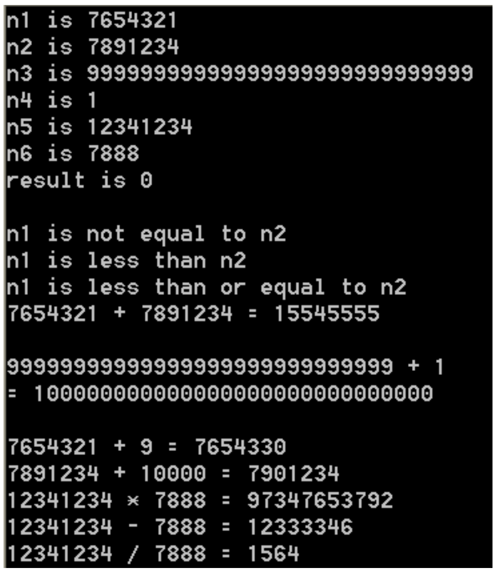
(HugeInt Class) A machine with 32-bit integers can represent integers in the range of approximately -2 billion to +2 billion. This fixed-size restriction is rarely troublesome, but there are applications in which we would like to be able to use a much wider range of integers. This is what C++ was built to do, namely, create powerful new data types. Study the class carefully, then answer the following:
a) Describe precisely how it operates.
b) What restrictions does the class have?
c) Overload the * multiplication operator.
d) Overload the / division operator.
e) Overload all the relational and equality operators.
[Note: We do not show an assignment operator or copy constructor for class HugeInteger, because the assignment operator and copy constructor provided by the compiler are capable of copying the entire array data member properly.]

2. 类定义参考

```
#ifndef HUGEINT_H
#define HUGEINT_H
```

```
#include <iostream>
using std::ostream;

class HugeInt
{
    friend ostream &operator<<( ostream &, const HugeInt & );
public:
    HugeInt( long = 0 ); // conversion/default constructor
    HugeInt( const char * ); // conversion constructor

    // addition operator; HugeInt + HugeInt
    HugeInt operator+( const HugeInt & ) const;

    // addition operator; HugeInt + int
    HugeInt operator+( int ) const;

    // addition operator;
    // HugeInt + string that represents large integer value
    HugeInt operator+( const char * ) const;

    bool operator==( const HugeInt & ) const; // equality operator
    bool operator!=( const HugeInt & ) const; // inequality operator
    bool operator<( const HugeInt & ) const; // less than operator

    // less than or equal to operator
    bool operator<=( const HugeInt & ) const;
    bool operator>( const HugeInt & ) const; // greater than operator

    // greater than or equal to operator
    bool operator>=( const HugeInt & ) const;
    HugeInt operator-( const HugeInt & ) const; // subtraction operator
    HugeInt operator*( const HugeInt & ) const; // multiply two HugeInts
    HugeInt operator/( const HugeInt & ) const; // divide two HugeInts

    int getLength() const;
private:
    int integer[ 40 ];
}; // end class HugeInt

#endif
```

3. 结果输出示例


```
n1 is 7654321
n2 is 7891234
n3 is 99999999999999999999999999999
n4 is 1
n5 is 12341234
n6 is 7888
result is 0

n1 is not equal to n2
n1 is less than n2
n1 is less than or equal to n2
7654321 + 7891234 = 15545555

99999999999999999999999999999 + 1
= 100000000000000000000000000000

7654321 + 9 = 7654330
7891234 + 10000 = 7901234
12341234 * 7888 = 97347653792
12341234 - 7888 = 12333346
12341234 / 7888 = 1564
```

作业三（习题 10.10，RationalNumber 类）

1. 问题描述

(RationalNumber Class) Create a class **RationalNumber** (fractions) with the following capabilities:

a) Create a constructor that prevents a 0 denominator in a fraction, reduces or simplifies fractions that are not in reduced form and avoids negative denominators.

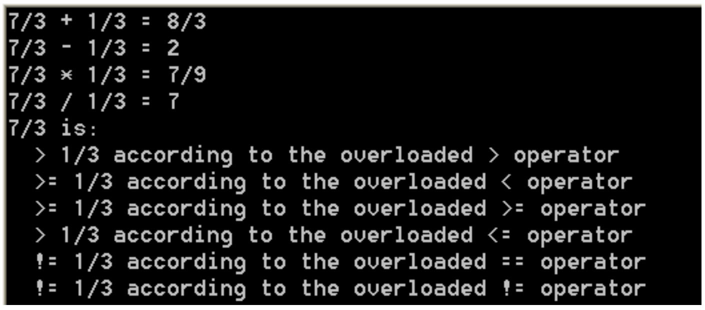
b) Overload the addition, subtraction, multiplication and division operators for this class.

c) Overload the relational and equality operators for this class.

2. 类定义参考

```
#ifndef RATIONAL_NUMBER_H
#define RATIONAL_NUMBER_H

class RationalNumber
{
public:
    RationalNumber( int = 0, int = 1 ); // default constructor
    RationalNumber operator+( const RationalNumber& ); // addition
    RationalNumber operator-( const RationalNumber& ); // subtraction
    RationalNumber operator*( const RationalNumber& ); // multiplication
    RationalNumber operator/( RationalNumber& ); // division

    // relational operators
    bool operator>( const RationalNumber& ) const;
    bool operator<( const RationalNumber& ) const;
    bool operator>=( const RationalNumber& ) const;
    bool operator<=( const RationalNumber& ) const;

    // equality operators
    bool operator==( const RationalNumber& ) const;
    bool operator!=( const RationalNumber& ) const;

    void printRational() const; // display rational number
private:
    int numerator; // private variable numerator
    int denominator; // private variable denominator
    void reduction(); // function for fraction reduction
}; // end class RationalNumber

#endif
```

3. 结果输出示例



```
7/3 + 1/3 = 8/3
7/3 - 1/3 = 2
7/3 * 1/3 = 7/9
7/3 / 1/3 = 7
7/3 is:
  > 1/3 according to the overloaded > operator
  >= 1/3 according to the overloaded < operator
  >= 1/3 according to the overloaded >= operator
  > 1/3 according to the overloaded <= operator
  != 1/3 according to the overloaded == operator
  != 1/3 according to the overloaded != operator
```

作业四（习题 10.11，Polynomial 类）

1. 问题描述

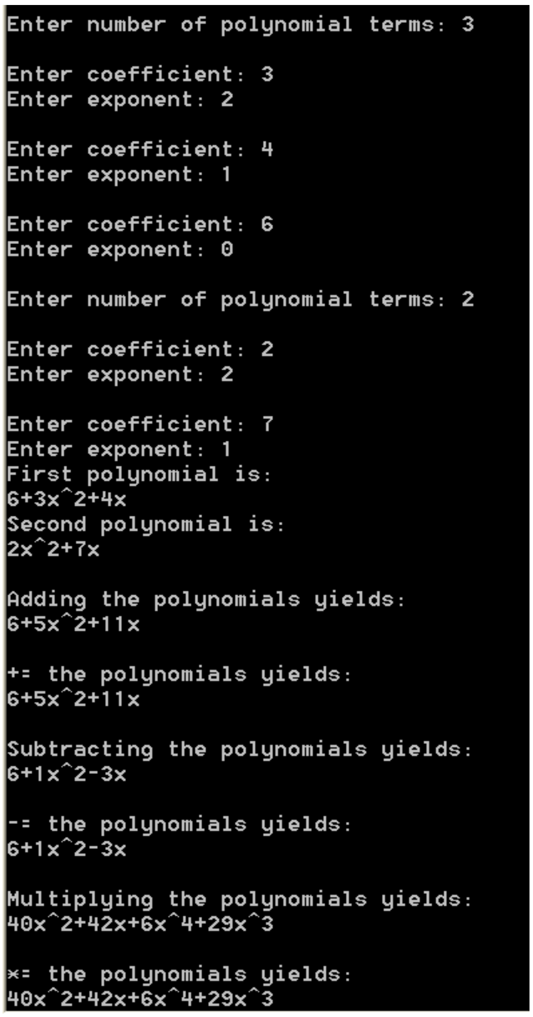
(Polynomial Class) Develop class **Polynomial**. The internal representation of a Polynomial is an array of terms. Each term contains a coefficient and an exponent, e.g., the term $2x^4$ has the coefficient 2 and the exponent 4. Develop a complete class containing proper constructor and destructor functions as well as set and get functions. The class should also provide the following overloaded operator capabilities:
a) Overload the addition operator (+) to add two Polynomials.
b) Overload the subtraction operator (-) to subtract two Polynomials.

c) Overload the assignment operator to assign one Polynomial to another.
d) Overload the multiplication operator (*) to multiply two Polynomials.
e) Overload the addition assignment operator (+=), subtraction assignment operator (-=), and multiplication assignment operator (*=).

2. 测试函数示例

```
int main()
{
    Polynomial a, b, c, t;

    a.enterTerms();
    b.enterTerms();
    t = a;      // save the value of a
    cout << "First polynomial is:\n";
    a.printPolynomial();
    cout << "Second polynomial is:\n";
    b.printPolynomial();
    cout << "\nAdding the polynomials yields:\n";
    c = a + b;
    c.printPolynomial();
    cout << "\n+= the polynomials yields:\n";
    a += b;
    a.printPolynomial();
    cout << "\nSubtracting the polynomials yields:\n";
    a = t;    // reset a to original value
    c = a - b;
    c.printPolynomial();
    cout << "\n-= the polynomials yields:\n";
    a -= b;
    a.printPolynomial();
    cout << "\nMultiplying the polynomials yields:\n";
    a = t;    // reset a to original value
    c = a * b;
    c.printPolynomial();
    cout << "\n*= the polynomials yields:\n";
    a *= b;
    a.printPolynomial();
    cout << endl;
    return 0;
} // end main
```

3. 结果输出示例



```
Enter number of polynomial terms: 3

Enter coefficient: 3
Enter exponent: 2

Enter coefficient: 4
Enter exponent: 1

Enter coefficient: 6
Enter exponent: 0

Enter number of polynomial terms: 2

Enter coefficient: 2
Enter exponent: 2

Enter coefficient: 7
Enter exponent: 1
First polynomial is:
6+3x^2+4x
Second polynomial is:
2x^2+7x

Adding the polynomials yields:
6+5x^2+11x

+= the polynomials yields:
6+5x^2+11x

Subtracting the polynomials yields:
6+1x^2-3x

-= the polynomials yields:
6+1x^2-3x

Multiplying the polynomials yields:
40x^2+42x+6x^4+29x^3

*= the polynomials yields:
40x^2+42x+6x^4+29x^3
```

作业五（习题 11.10，Account 类）

1. 问题描述

创建一个与银行账户相关的类继承层次。银行的所有账户都可以存款和取款。存款能够产生一定的利息，查询和取款交易要缴纳一定的手续费。

要求：	基类	Account（参考实验提示）
	派生类	SavingAccount 和 CheckingAccount

- SavingAccount：
  - 继承 Account 的成员函数
  - 构造函数接收两个参数：存款初始值 initialBalance 和利率 rate
  - 增加一个数据成员：利率 interestRate

    - 增加 public 类型的成员函数用于计算利率 calculateInterest()
- CheckingAccount：
    - 构造函数应接收到两个参数，一个是存款初始值 initialBalance，一个是手续费 fee
    - 增加一个数据成员：手续费 transactionFee
    - 重新定义成员函数 credit()和 debit()，以便能够从存款余额中减去手续费，要求成员函数通过调用基类的成员函数来更新存款数目，debit()函数应该在确定取款之后才能扣除手续费

2. 实验提示

1）类定义示例：

```
#ifndef ACCOUNT_H
#define ACCOUNT_H

class Account
{
public:
    Account( double ); // constructor initializes balance
    void credit( double ); // add an amount to the account balance
    bool debit( double ); // subtract an amount from the account balance
    void setBalance( double ); // sets the account balance
    double getBalance(); // return the account balance
private:
    double balance; // data member that stores the balance
}; // end class Account

#endif

// Exercise 12.10 Solution: Account.cpp
// Member-function definitions for class Account.
#include <iostream>
using std::cout;
using std::endl;

#include "Account.h" // include definition of class Account

// Account constructor initializes data member balance
Account::Account( double initialBalance )
{
    // if initialBalance is greater than or equal to 0.0, set this value
    // as the balance of the Account
    if ( initialBalance >= 0.0 )
        balance = initialBalance;
    else // otherwise, output message and set balance to 0.0
    {
        cout << "Error: Initial balance cannot be negative." << endl;
        balance = 0.0;
    } // end if...else
} // end Account constructor

// credit (add) an amount to the account balance
void Account::credit( double amount )
{
    balance = balance + amount; // add amount to balance
} // end function credit

// debit (subtract) an amount from the account balance
// return bool indicating whether money was debited
bool Account::debit( double amount )
{
    if ( amount > balance ) // debit amount exceeds balance
    {
        cout << "Debit amount exceeded account balance." << endl;
        return false;
    } // end if
    else // debit amount does not exceed balance
    {
        balance = balance - amount;
        return true;
    } // end else
} // end function debit

// set the account balance
void Account::setBalance( double newBalance )
{
    balance = newBalance;
} // end function setBalance

// return the account balance
double Account::getBalance()
{
    return balance;
} // end function getBalance
```

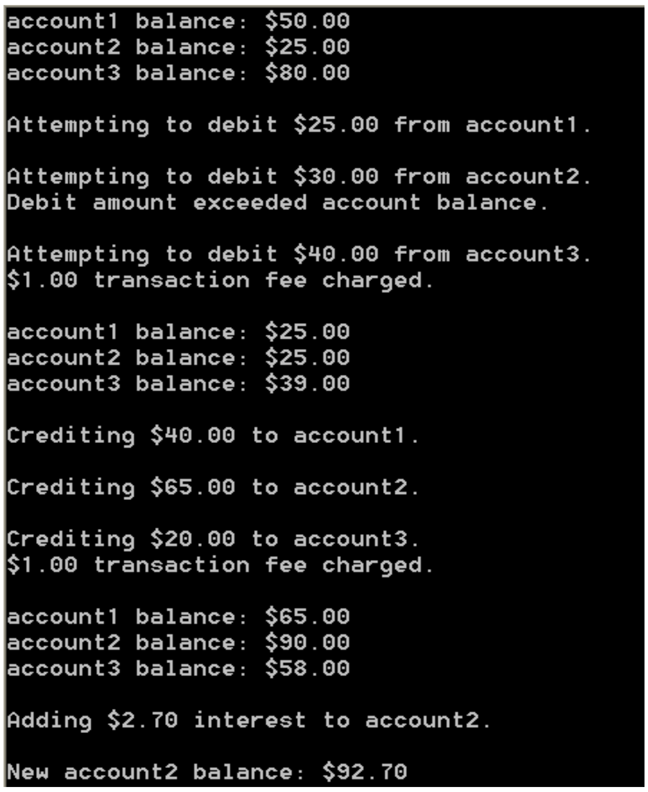
2）测试函数示例

```
int main()
{
```

```
Account account1( 50.0 ); // create Account object
SavingsAccount account2( 25.0, .03 ); // create SavingsAccount object
CheckingAccount account3( 80.0, 1.0 ); // create CheckingAccount object

cout << fixed << setprecision( 2 );

// display initial balance of each object
cout << "account1 balance: $" << account1.getBalance() << endl;
cout << "account2 balance: $" << account2.getBalance() << endl;
cout << "account3 balance: $" << account3.getBalance() << endl;

cout << "\nAttempting to debit $25.00 from account1." << endl;
account1.debit( 25.0 ); // try to debit $25.00 from account1
cout << "\nAttempting to debit $30.00 from account2." << endl;
account2.debit( 30.0 ); // try to debit $30.00 from account2
cout << "\nAttempting to debit $40.00 from account3." << endl;
account3.debit( 40.0 ); // try to debit $40.00 from account3

// display balances
cout << "\naccount1 balance: $" << account1.getBalance() << endl;
cout << "account2 balance: $" << account2.getBalance() << endl;
cout << "account3 balance: $" << account3.getBalance() << endl;

cout << "\nCrediting $40.00 to account1." << endl;
account1.credit( 40.0 ); // credit $40.00 to account1
cout << "\nCrediting $65.00 to account2." << endl;
account2.credit( 65.0 ); // credit $65.00 to account2
cout << "\nCrediting $20.00 to account3." << endl;
account3.credit( 20.0 ); // credit $20.00 to account3

// display balances
cout << "\naccount1 balance: $" << account1.getBalance() << endl;
cout << "account2 balance: $" << account2.getBalance() << endl;
cout << "account3 balance: $" << account3.getBalance() << endl;

// add interest to SavingsAccount object account2
double interestEarned = account2.calculateInterest();
cout << "\nAdding $" << interestEarned << " interest to account2."
    << endl;
account2.credit( interestEarned );

cout << "\nNew account2 balance: $" << account2.getBalance() << endl;
return 0;
```

```
} // end main
```

3. 结果示例



```
account1 balance: $50.00
account2 balance: $25.00
account3 balance: $80.00

Attempting to debit $25.00 from account1.

Attempting to debit $30.00 from account2.
Debit amount exceeded account balance.

Attempting to debit $40.00 from account3.
$1.00 transaction fee charged.

account1 balance: $25.00
account2 balance: $25.00
account3 balance: $39.00

Crediting $40.00 to account1.

Crediting $65.00 to account2.

Crediting $20.00 to account3.
$1.00 transaction fee charged.

account1 balance: $65.00
account2 balance: $90.00
account3 balance: $58.00

Adding $2.70 interest to account2.

New account2 balance: $92.70
```