学号 E21714049 专业 计算机科学与技术 姓名 梅世祺
实验日期 2019.05.23 教师签字 成绩

# 实验报告

【实验名称】运算符重载

【实验目的】

1. 了解运算符重载的实际意义。
2. 掌握运算符重载的规则。
3. 熟悉多种运算符重载的实际应用。

【实验原理】

运算符重载实质上是函数重载，重载为成员函数，它就可以自由地访问本类的数据成员。实际使用时，总是通过该类的某个对象来访问重载的运算符。

【实验内容】

## 实验一 复数运算

**题目：**定义复数类 Complex，利用运算符重载实现复数的加、减、乘、除四则运算。

**要求：**1. 复数类包括实部（real）和虚部（image）两个数据成员，可以分别读取、设置和输出。

2. 每个运算测试 3 组数据。

3. 注意程序的健壮性，如除零操作。

**原理：** 1. 加法规则，(a+bi)+(c+di)=(a+c)+(b+d)i。

2. 减法规则，(a+bi)-(c+di)=(a-c)+(b-d)i。

3. 乘法规则，(a+bi)*(c+di)=(ac-bd)+(ad+bc)i。

**实验结果（含源码）：**

Complex 类声明如下：

```
class Complex {
    public:
        Complex(double real, double image) : real(real), image(image) {};

        Complex operator+ (const Complex &c2) const;
        Complex operator- (const Complex &c2) const;
        Complex operator* (const Complex &c2) const;
        Complex operator/ (const Complex &c2) const;

        double getReal();
        double getImage();
        void setReal(double real);
        void setImage(double image);
    private:
        double real, image;
};
```

针对加减乘除的运算符重载实现如下：

```
Complex Complex::operator+ (const Complex &c2) const {
    return Complex(real + c2.real, image + c2.image);
}

Complex Complex::operator- (const Complex &c2) const {
    return Complex(real - c2.real, image - c2.image);
}

Complex Complex::operator* (const Complex &c2) const {
    return Complex(
        real * c2.real - image * c2.image,
        real * c2.image + image * c2.real
    );
}

Complex Complex::operator/ (const Complex &c2) const {
    double denominator = c2.real * c2.real + c2.image * c2.image;

    if (denominator <= 0) {
        throw "\n\033[31m[ZeroDivisionError]\033[0m Divisor cannot be 0!";
    }

    return Complex(
        (real * c2.real + image * c2.image) / denominator,
        (image * c2.real - real * c2.image) / denominator
    );
}
```

其它一些普通 getter 方法和 setter 方法如下：

```
double Complex::getReal() {
    return real;
}

double Complex::getImage() {
    return image;
}

void Complex::setReal(double real) {
    Complex::real = real;
}

void Complex::setImage(double  image) {
    Complex::image = image;
}
```

它们分别用来访问和设置复数对象的实部和虚部。

主函数测试代码如下：

```
int main(int argc, char const *argv[])
{
    Complex c1(1, 1), c2(2, 3), c3(0, 0);

    cout<<printComplex(c1)<<"+"<<printComplex(c2)<<"="<<printComplex(c1+c2)<<endl;
    cout<<printComplex(c1)<<"+"<<printComplex(c3)<<"="<<printComplex(c1+c3)<<endl;
    cout<<printComplex(c2)<<"+"<<printComplex(c3)<<"="<<printComplex(c2+c3)<<endl<<endl;

    cout<<printComplex(c1)<<"-"<<printComplex(c2)<<"="<<printComplex(c1-c2)<<endl;
    cout<<printComplex(c1)<<"-"<<printComplex(c3)<<"="<<printComplex(c1-c3)<<endl;
    cout<<printComplex(c2)<<"-"<<printComplex(c3)<<"="<<printComplex(c2-c3)<<endl<<endl;

    cout<<printComplex(c1)<<"*"<<printComplex(c2)<<"="<<printComplex(c1*c2)<<endl;
    cout<<printComplex(c1)<<"*"<<printComplex(c3)<<"="<<printComplex(c1*c3)<<endl;
    cout<<printComplex(c2)<<"*"<<printComplex(c3)<<"="<<printComplex(c2*c3)<<endl<<endl;

    cout<<printComplex(c1)<<"/"<<printComplex(c2)<<"="<<printComplex(c1/c2)<<endl;
    cout<<printComplex(c3)<<"/"<<printComplex(c2)<<"="<<printComplex(c3/c2)<<endl;
    try {
        cout<<printComplex(c2)<<"/"<<printComplex(c3)<<"="<<printComplex(c2/c3)<<endl;
    } catch (const char* error) {
        cout<<error<<endl;
    }
    return 0;
}
```

程序运行结果：

```
----------  构建用时:180 ms --------------
(1.000000,1.000000)+(2.000000,3.000000)=(3.000000,4.000000)
(1.000000,1.000000)+(0.000000,0.000000)=(1.000000,1.000000)
(2.000000,3.000000)+(0.000000,0.000000)=(2.000000,3.000000)

(1.000000,1.000000)-(2.000000,3.000000)=(-1.000000,-2.000000)
(1.000000,1.000000)-(0.000000,0.000000)=(1.000000,1.000000)
(2.000000,3.000000)-(0.000000,0.000000)=(2.000000,3.000000)

(1.000000,1.000000)*(2.000000,3.000000)=(-1.000000,5.000000)
(1.000000,1.000000)*(0.000000,0.000000)=(0.000000,0.000000)
(2.000000,3.000000)*(0.000000,0.000000)=(0.000000,0.000000)

(1.000000,1.000000)/(2.000000,3.000000)=(0.384615,-0.076923)
(0.000000,0.000000)/(2.000000,3.000000)=(0.000000,0.000000)
(2.000000,3.000000)/(0.000000,0.000000)=
[ZeroDivisionError] Divisor cannot be 0!
```

# 实验二 时间运算

**题目：**设计并实现一个时间类及其相关运算。

**要求：**1. 时间类包括时、分、秒。

2. 可以读取和设置时、分、秒。

3. 可以输出格式化时间 HH：MM：SS。

4. 实现时间比较运算，大于（>）、小于（<）和等于（==）。

5. 实现时间算术运算，相加（+）、相减（-）、自增一秒（++前、后置）、自减一秒（--前、后置）增加 n 秒（+=）、减少 n 秒（-=）。

6. 每个运算符测试 3 组数据。

7. 注意程序的健壮性。

**原理：**

**提高（选做）：**设计并实现日期时间类（包括时间和日期）及其相关运算，包括比较运算和算术运算，注意每个月的天数不同，注意闰年问题。

**实验结果（含源码）：**

MyTime 类声明如下：

```
class MyTime {
    public:
        MyTime(int hr, int min, int sec) {
            if (hr < 0 || min < 0 || sec < 0) {
                throw "\n\033[31mInvalid time format!\033[0m";
            }

            hour = hr;
            minute = min;
            second = sec;
        }

        bool operator> (const MyTime &mt2) const;
        bool operator< (const MyTime &mt2) const;
        bool operator== (const MyTime &mt2) const;
        MyTime operator+ (const MyTime &mt2) const;
        MyTime operator- (const MyTime &mt2) const;
        MyTime& operator++ ();
        MyTime operator++ (int);
        MyTime& operator-- ();
        MyTime operator-- (int);
        MyTime operator+= (const int &n);
        MyTime operator-= (const int &n);

        int getHour();
        int getMinute();
        int getSecond();

        void setHour(int hour);
        void setMinute(int minute);
        void setSecond(int second);
        void print();
    private:
        int hour, minute, second;
};
```

针对比较运算符重载如下：

```cpp
bool MyTime::operator> (const MyTime &mt2) const {
    if (hour > mt2.hour) {
        return true;
    }
    if (minute > mt2.minute) {
        return true;
    }
    if (second > mt2.second) {
        return true;
    }

    return false;
}

bool MyTime::operator< (const MyTime &mt2) const {
    if (hour < mt2.hour) {
        return true;
    }
    if (minute < mt2.minute) {
        return true;
    }
    if (second < mt2.second) {
        return true;
    }

    return false;
}

bool MyTime::operator== (const MyTime &mt2) const {
    if (hour == mt2.hour && minute == mt2.minute && second == mt2.second) {
        return true;
    }

    return false;
}
```

针对加减运算符的重载实现如下：

```
MyTime MyTime::operator+ (const MyTime &mt2) const {
    int totalSecondsA = hour * 3600 + minute * 60 + second;
    int totalSecondsB = mt2.hour * 3600 + mt2.minute * 60 + mt2.second;
    int result = totalSecondsA + totalSecondsB;

    int resultHour = result / 3600;
    int resultMinute = (result % 3600) / 60;
    int resultSecond = result % 3600 % 60;
    return MyTime(resultHour, resultMinute, resultSecond);
}

MyTime MyTime::operator- (const MyTime &mt2) const {
    int totalSecondsA = hour * 3600 + minute * 60 + second;
    int totalSecondsB = mt2.hour * 3600 + mt2.minute * 60 + mt2.second;
    int result = totalSecondsA - totalSecondsB;

    int resultHour = result / 3600;
    int resultMinute = (result % 3600) / 60;
    int resultSecond = result % 3600 % 60;
    return MyTime(resultHour, resultMinute, resultSecond);
}
```

这里在进行时间的加减时，采用了全部转化为秒来进行运算的思路。

针对自增自减运算符的重载实现如下:

```
MyTime& MyTime::operator++ () {
    second++;
    if (second>=60) {
        second-=60;
        minute++;
        if (minute>=60) {
            hour++;
        }
    }

    return * this;
}

MyTime MyTime::operator++ (int) {
    MyTime old = * this;
    ++ (* this);
    return old;
}

MyTime& MyTime::operator-- () {
    second--;
    if (second<0) {
        second+=60;
        minute--;
        if (minute<0) {
            hour--;
            if (hour<0) {
                throw "\n\033[31mInvalid Operation\033[0m";
            }
        }
    }

    return * this;
}
```

```
MyTime MyTime::operator-- (int) {
    MyTime old = * this;
    -- (* this);
    return old;
}
```

针对 +=、-= 的运算符重载实现如下：

```
MyTime MyTime::operator+= (const int &n) {
    second += n;
    if (second>=60) {
        second-=60;
        minute++;
        if (minute>=60) {
            hour++;
        }
    }

    return * this;
}

MyTime MyTime::operator-= (const int &n) {
    second-=n;
    if (second<0) {
        second+=60;
        minute--;
        if (minute<0) {
            hour--;
            if (hour<0) {
                throw "\n\033[31mInvalid Operation\033[0m";
            }
        }
    }

    return * this;
}
```

对于 setter 函数和 getter 成员函数的实现如下：

```
int MyTime::getHour() {
    return hour;
}

int MyTime::getMinute() {
    return minute;
}

int MyTime::getSecond() {
    return second;
}

void MyTime::setHour(int hour) {
    MyTime::hour = hour;
}

void MyTime::setMinute(int minute) {
    MyTime::minute = minute;
}

void MyTime::setSecond(int second) {
    MyTime::second = second;
}
```

以及其它一些普通成员函数的实现如下：

```
void MyTime::print() {
    std::cout<<hour<<":";
    if (minute < 10) {
        std::cout<<"0"<<minute;
    } else {
        std::cout<<minute;
    }
    std::cout<<":";
    if (second < 10) {
        std::cout<<"0"<<second;
    } else {
        std::cout<<second;
    }

}
```

主函数的测试代码实现如下：

```
int main(int argc, char const *argv[])
{
    MyTime t1(1,1,1), t2(12,12,12), t3(59,59,59);
    int INTERVAL = 30;

    t1.print(); cout<<" + "; t2.print(); cout<<" = "; (t1+t2).print(); cout<<endl;
    t1.print(); cout<<" + "; t3.print(); cout<<" = "; (t1+t3).print(); cout<<endl;
    t2.print(); cout<<" + "; t3.print(); cout<<" = "; (t2+t3).print(); cout<<endl;

    try {
        t1.print(); cout<<" - "; t2.print(); cout<<" = "; (t1-t2).print(); cout<<endl;
    } catch (const char* msg) {
        cout<<msg<<endl;
    }
    try {
        t1.print(); cout<<" - "; t3.print(); cout<<" = "; (t1-t3).print(); cout<<endl;
    } catch (const char* msg) {
        cout<<msg<<endl;
    }
    try {
        t3.print(); cout<<" - "; t2.print(); cout<<" = "; (t3-t2).print(); cout<<endl;
    } catch (const char* msg) {
        cout<<msg<<endl;
    }
```

```
    t1.print(); cout<<" == "; t1.print(); cout<<" "<<(t1 == t1 ? "true" : "false")<<endl;
    t1.print(); cout<<" == "; t2.print(); cout<<" "<<(t1 == t2 ? "true" : "false")<<endl;
    t2.print(); cout<<" == "; t3.print(); cout<<" "<<(t2 == t3 ? "true" : "false")<<endl;

    t1.print(); cout<<" > "; t1.print(); cout<<" "<<(t1 > t1 ? "true" : "false")<<endl;
    t1.print(); cout<<" > "; t2.print(); cout<<" "<<(t1 > t2 ? "true" : "false")<<endl;
    t3.print(); cout<<" > "; t2.print(); cout<<" "<<(t3 > t2 ? "true" : "false")<<endl;

    t1.print(); cout<<" < "; t1.print(); cout<<" "<<(t1 < t1 ? "true" : "false")<<endl;
    t1.print(); cout<<" < "; t2.print(); cout<<" "<<(t1 < t2 ? "true" : "false")<<endl;
    t3.print(); cout<<" < "; t2.print(); cout<<" "<<(t3 < t2 ? "true" : "false")<<endl;

    cout<<"++"; t1.print(); cout<<" = "; (++t1).print(); cout<<endl;
    cout<<"--"; t1.print(); cout<<" = "; (--t1).print(); cout<<endl;
    t1.print(); cout<<"++ = "; (t1++).print(); cout<<endl;
    t1.print(); cout<<"-- = "; (t1--).print(); cout<<endl;

    t1.print(); cout<<" += "<<INTERVAL<<" = "; (t1+=INTERVAL).print(); cout<<endl;
    t1.print(); cout<<" += "<<INTERVAL/2<<" = "; (t1+=INTERVAL/2).print(); cout<<endl;
    t1.print(); cout<<" += "<<INTERVAL*2<<" = "; (t1+=INTERVAL*2).print(); cout<<endl;

    t1.print(); cout<<" -= "<<INTERVAL<<" = "; (t1-=INTERVAL).print(); cout<<endl;
    t1.print(); cout<<" -= "<<INTERVAL/2<<" = "; (t1-=INTERVAL/2).print(); cout<<endl;
    t1.print(); cout<<" -= "<<INTERVAL*2<<" = "; (t1-=INTERVAL*2).print(); cout<<endl;

    return 0;
}
```

程序运行结果如下：

```
----------  构建用时:409 ms --------------
1:01:01 + 12:12:12 = 13:13:13
1:01:01 + 59:59:59 = 61:01:00
12:12:12 + 59:59:59 = 72:12:11
1:01:01 - 12:12:12 =
Invalid time format!
1:01:01 - 59:59:59 =
Invalid time format!
59:59:59 - 12:12:12 = 47:47:47
1:01:01 == 1:01:01 true
1:01:01 == 12:12:12 false
12:12:12 == 59:59:59 false
1:01:01 > 1:01:01 false
1:01:01 > 12:12:12 false
59:59:59 > 12:12:12 true
1:01:01 < 1:01:01 false
1:01:01 < 12:12:12 true
59:59:59 < 12:12:12 false
++1:01:01 = 1:01:02
--1:01:02 = 1:01:01
1:01:01++ = 1:01:01
1:01:02-- = 1:01:02
1:01:01 += 30 = 1:01:31
1:01:31 += 15 = 1:01:46
1:01:46 += 60 = 1:02:46
1:02:46 -= 30 = 1:02:16
1:02:16 -= 15 = 1:02:01
1:02:01 -= 60 = 1:01:01
➜
```