Assignment 4

191220022 丁一凡

一、 概念题

1.1 简述Demeter法则的基本思想；过度使用Demeter法则会带来什么问题

Demeter法则：一个类的成员函数除了能访问自身类结构的直接子结构外，不能以任何方式依赖于其他类的子结构；并且每个成员函数只应向某个有限集合中的对象发送消息。

过度使用Demeter法则会导致各模块过于独立，从而导致访问数据时必须要通过给定的有限的接口访问，增大访问时的开销，不易于快速地访问。

1.2 简述为什么要对操作符进行重载；操作符重载会带来什么问题

可以对已有的操作符重载，从而使其可以对自定义类型（类）的对象进行操作，使其对应的操作更加直观，同时提高程序设计语言的灵活性，给程序设计带来方便

操作符重载后如果与原操作符语义有歧义，使用时会产生错误

1.3 简述操作符重载的两种形式；这两种形式有什么区别

操作符重载的两种形式：

① 作为一个类的非静态的成员函数

② 作为一个全局（友元）函数

作为成员函数时，因为其隐含的 `this` 指针，所以只需一个参数，而使用全局（友元）函数时，因其不包含 `this` 指针，所以需要显式指出所有参数。同时也正是由于这一点，部分操作符的重载只能通过全局（友元）函数实现。

二、 编程题

2.1

```
class Datetime;
class Date
{
private:
    int _year, _month, _day;
public:
    Date(int year, int month, int day)
```

```
    {
        _year = year, _month = month, _day = day;
    }
    bool is_leapyear()
    {
        if((_year % 4 ==0) && (_year % 100 != 0) && (_year % 400 ==0))
            return true;
        else
            return false;
    }
    bool is_31days()
    {
        if(_month == 1 || _month == 3 || _month == 5 || _month == 7 || _month ==
8 || _month == 10 || _month == 12)
            return true;
        else
            return false;

    }
    bool operator ==(const Date& d) const
    {
        return (_year == d._year) && (_month == d._month) && (_day == d._day);
    }
    bool operator <(const Date& d) const
    {
        if(_year < d._year)
            return true;
        else if(_year > d._year)
            return false;
        if(_month < d._month)
            return true;
        else if(_month > d._month)
            return false;
        if(_day < d._day)
            return true;
        else
            return false;
    }
    friend class Datetime;
};

class Time
{
private:
    int _hour, _minute, _second;
public:
    Time(int hour, int minute, int second)
    {
        _hour = hour, _minute = minute, _second = second;
    }
    bool operator ==(const Time& t) const
    {
        return (_hour == t._hour) && (_minute == t._minute) && (_second ==
t._second);
    }
    bool operator <(const Time& t) const
    {
        if(_hour < t._hour)
```

```
            return true;
        else if(_hour > t._hour)
            return false;
        if(_minute < t._minute)
            return true;
        else if(_minute > t._minute)
            return false;
        if(_second < t._second)
            return true;
        else
            return false;
    }
    friend class Datetime;
};


class Datetime
{
private:
    Date _date;
    Time _time;
public:
    Datetime(const Date& date, const Time& time);
    std::string to_string() const;
    Datetime operator + (long s) const
    {
        _time.second = ((long)_time.second + s) % 60;
        long min = _time.minute + (_time.second+s)/60;
        _time.minute = min % 60;
        long hour = _time.hour + min/60;
        _time.hour = hour%24;
        long day = _date.day + hour/24;
        _date.day = day%30;
        long month = _data.month + day/30;
        _date.month = month%12;
        year = year + month / 12;

    }
    Datetime operator ++()
    {
        (*this) = (*this) + 1;
        return (*this);
    }
    Datetime operator --()
    {
        (*this) = (*this) - 1;
        return (*this);
    }
    bool operator < (const Datetime& d) const
    {
        if(_date < d._date)
            return true;
        else if(_date > d._date)
            return false;
        if(_time < d._time)
            return true;
        else
            return false;
```

```
    }
    bool operator == (const Datetime& d) const
    {
        return (this->_date == d._date) && (this->_time == d._time);
    }
};
```