Assignment 4

Fuyun Wang

191300051

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法则的基本思想：

- 每个单元对于其他的单元只能拥有有限的知识：只是与当前单元紧密联系的单元
- 每个单元只能和它的朋友交谈：不能和陌生单元交谈
- 只和自己直接的朋友交谈

一个类的成员函数除了能够访问自身类结构的直接子结构外，不能够以任何方式依赖于其它类的结构。

过度使用Demeter法则的问题：

过度使用迪米特法则，也会造成系统的不同模块之间的通信效率降低，使系统的不同模块之间不容易协调等缺点。同时，因为迪米特法则要求类与类之间尽量不直接通信，如果类之间需要通信就通过第三方转发的方式，这就直接导致了系统中存在大量的中介类，这些类存在的唯一原因是为了传递类与类之间的相互调用关系，这就毫无疑问的增加了系统的复杂度。

1.2 简述为什么要对操作符重载进行重载；操作符重载会带来什么问题。

为什么？

以负数的实现中，直接使用函数命名的方式不符合平时书写的习惯，C++允许对已有的操作符进行重载，使得他们能够对自定义类型的对象进行操作。

与函数名重载一样，操作符重载也是实现多态性的一种语言机制。

- 简单、方便，比起封装一个函数，使用operator比较形象直观
- 可以提高代码的可读性
- 为了处理自定义类型和内置类型之间的运算

会带来什么问题？

- 不合适的操作符重载会造成运算的混乱或者错误。
- 可能会对原有的定义产生覆盖或者干扰

注意点：

1. 不可臆造运算符。
2. 运算符原有操作数的个数、优先级和结合性不能改变。
3. 操作数中至少一个是自定义类型。
4. 保持重载运算符的自然含义。

1.3 简述操作符重载的两种形式；这两种形式有什么区别。

两种形式

- 作为一个类的非静态的成员函数
- 作为一个全局（友元函数）

区别

一般情况下操作符既可以作为全局操作符也可以当作成员函数进行重载。在有些情况下，操作符则只能作为全局函数或者只能作为类的成员函数进行重载。

以复数的加减为例（当double类型的数据与自定义类型的数据进行加减时），只能够使用全局函数重载而不能够使用类的成员函数重载。

一般情况下，对于单目运算符，我们常常都是重载为成员函数。

对于双目操作符

对于双目运算符:

a.当重载为成员函数时，会隐含一个this指针；当重载为友元函数时，将不存在隐含的this指针，需要在参数列表中显示地添加操作数。
b.当重载为成员函数时，只允许右参数的隐式转换；当重载为友元函数时，能够接受左参数和右参数的隐式转换。

二

```cpp
//Timer.cpp
#include <iostream>
using namespace std;

#define MAX_SECOND 60
#define MAX_MINUTE 60
#define MAXhour 24
#define MAXmonth 12

#define IS_LMONTH(month) (month == 1 || month == 3 || month == 5 || month == 7 || month == 8 || month == 10 || month == 12)

class Date
{
private:
    int year, month, day;
    friend class Datetime;
```

```
17
18  public:
19      Date(int year, int month, int day)
20      {
21          this->year = year;
22          this->month = month;
23          this->day = day;
24      }
25  };
26  class Time
27  {
28  private:
29      int hour, minute, second;
30      friend class Datetime;
31  public:
32      Time(int hour, int minute, int second)
33      {
34          this->hour = hour;
35          this->minute = minute;
36          this->second = second;
37      }
38  };
39  class Datetime
40  {
41  private:
42      Date date;
43      Time time;
44      int maxday;
45      bool is_leap_yaer() const
46      {
47          return ((date.year % 4 == 0) && (date.year % 100 != 0)) ||
    (date.year % 400 == 0);
48      }
49      void set_maxday()
50      {
51          if (IS_LMONTH(date.month))
52          {
53              maxday = 31;
54          }
55          else if (date.month == 2)
56          {
57              if (is_leap_yaer())
58                  maxday = 29;
59              else
60                  maxday = 28;
61          }
62          else
63          {
64              maxday = 30;
```

```
65          }
66      }
67      void simp()
68      {
69          while (time.second >= MAX_SECOND)
70          {
71              time.second -= MAX_SECOND;
72              time.minute += 1;
73          }
74          while (time.second < 0)
75          {
76              time.second += MAX_SECOND;
77              time.minute -= 1;
78          }
79          while (time.minute >= MAX_MINUTE)
80          {
81              time.minute -= MAX_MINUTE;
82              time.hour += 1;
83          }
84          while (time.minute < 0)
85          {
86              time.minute += MAX_MINUTE;
87              time.hour -= 1;
88          }
89          while (time.hour >= MAXhour)
90          {
91              time.hour -= MAXhour;
92              date.day += 1;
93          }
94          while (time.hour < 0)
95          {
96              time.hour += MAXhour;
97              date.day -= 1;
98          }
99          while (date.day > maxday)
100         {
101             date.day -= maxday;
102             date.month += 1;
103             set_maxday();
104         }
105         while (date.day <= 0)
106         {
107             date.day += maxday;
108             date.month -= 1;
109             set_maxday();
110         }
111         while (date.month > MAXmonth)
112         {
113             date.month -= MAXmonth;
```

```
114             date.year += 1;
115         }
116         while (date.month <= 0)
117         {
118             date.month += MAXmonth;
119             date.year -= 1;
120         }
121     }
122
123 public:
124     std::string to_string() const
125     {
126         char tmp[256];
127         printf("%04d-%02d-%02d|%02d:%02d:%02d\n", date.year,
    date.month, date.day, time.hour, time.minute, time.second);
128         sscanf(tmp, "%04d-%02d-%02d|%02d:%02d:%02d\n", date.year,
    date.month, date.day, time.hour, time.minute, time.second);
129         string ans(tmp);
130         return ans;
131     }
132     Datetime(const Date &date, const Time &time) : date(date),
    time(time)
133     {
134         set_maxday();
135         simp();
136     }
137     Datetime operator+(int sec)
138     {
139
140         //如果你是在函数中创建了一个临时对象，你是不能够返回它的引用或者指针的。
    因为它会被清理掉。一定是返回的值。
141         //但是如果返回的是参数当中传进来的引用，那么可以直接返回引用。如=的操作
    符重载。
142         Datetime temp(*this);
143         temp.time.second += sec;
144         temp.simp();
145         return temp;
146     }
147     Datetime operator-(int sec)
148     {
149         Datetime temp(*this);
150         temp.time.second-=sec;
151         temp.simp();
152         return temp;
153     }
154     bool operator<(const Datetime &dt)
155     {
156         if (date.year<dt.date.year) return true;
157         else if (date.year>dt.date.year) return false;
```

```
158         else
159         {
160             if (date.month<dt.date.month) return true;
161             else if (date.month>dt.date.month) return false;
162             else
163             {
164                 if (date.day<dt.date.day) return true;
165                 else if (date.day>dt.date.day) return false;
166                 else
167                 {
168                     if (time.hour<dt.time.hour) return true;
169                     else if (time.hour>dt.time.hour) return false;
170                     else
171                     {
172                         if (time.minute<dt.time.minute) return true;
173                         else if(time.minute>dt.time.minute)return
    false;
174                         else
175                         {
176                             if (time.second<dt.time.second) return
    true;
177                             else return false;
178                         }
179                     }
180                 }
181             }
182         }
183     }
184     bool operator==(const Datetime &dt)
185     {
186         return date.year==dt.date.year && date.month==dt.date.month
    && date.day==dt.date.day && time.hour==dt.time.hour &&
    time.minute==dt.time.minute && time.second==dt.time.second;
187     }
188     double operator-(const Datetime& dt)
189     {
190         int ans=0;
191         if (*this<dt)
192         {
193             while (((*this)==dt)==0)
194             {
195                 (*this)++;
196                 ans--;
197             }
198             return ans;
199         }
200         else if (*this==dt)
201         {
202             return ans;
```

```
203         }
204         else
205         {
206             while (((*this)==dt)==0)
207             {
208                 (*this)--;
209                 ans++;
210             }
211             return ans;
212         }
213     }
214 
215     void operator++()
216     {
217         time.second++;
218         simp();
219     }
220     void operator++(int)
221     {
222         time.second++;
223         simp();
224     }
225     void operator--()
226     {
227         time.second--;
228         simp();
229     }
230     void operator--(int)
231     {
232         time.second--;
233         simp();
234     }
235 
236 
237 };
238 
239 int main()
240 {
241     Date date1(2001, 1, 26);
242     Time time1(0, 0, 0);
243     Datetime datetime1(date1, time1);
244     datetime1.to_string();
245     datetime1--;
246     datetime1.to_string();
247     Datetime datetime2=datetime1+1;
248     datetime2.to_string();
249     cout<<datetime2-datetime1<<endl;
250     for (int i=0;i<10000;i++)
251     {
```

```
252             datetime1++;
253             datetime1.to_string();
254         }
255     }
```