高级程序设计 2020 春

实验报告

指导老师：陈家骏 黄书剑 姓名：王晨渊, 学号：181220057

1 概念题

1.1 C++ 中操作符的重载遵循哪些基本原则？

只能重载 C++ 语言中已有的操作符，不可臆造新的操作符

可以重载 C++ 中除. .* ?: :: sizeof 外的所有操作符

需要遵循已有操作符的语法，一方面不能改变操作数的个数，另一方面不能改变原操作符的优先级和结合性。

尽量遵循已有操作符原来的语义。语言本身没有对此做任何规定，使用者需要自己把握。

1.2 简述单目运算符 (++，–) 前置重载和后置重载的差别。

如果不做特殊处理，前置和后置共用同一个重载函数，但失去了原先的语义。

前置为左值表达式，后置为右值表达式。

因此，定义前置的自增（自减）操作符时，函数原型为

< 返回值类型 > operator #(); 其中返回值为引用类型。

定义后置的自增（自减）操作符时，函数原型为

< 返回值类型 > operator #(int); 其中返回值为常量类型。

2 编程题

2.1 修改原程序

原程序不能正常运行。

由于在类中没有显式地重载 = 操作符，a2=a1 调用的是 C++ 为其创建的隐式默认重载函数，默认将 a1 的所有成员原样赋值给 a2。这样 a2.p 和 a1.p 指向了相同的内存空间。当 a1 或 a2 其中一个消亡，另一个没有消亡的时候，没有消亡的对象对于 p 指向的内容的访问会出现问题。并且 a1 和 a2 都消亡时，指向的空间被释放两次引发报错。

应该将其修改为以下内容。

```
1  #include <iostream>
2  using namespace std;
3  class A
```

```
4       {
5       public:
6           int x;
7           int *p;
8           A()
9           {
10              p = new int(0);
11              x = 0;
12          }
13
14          A& operator = (const A& a)
15          {
16              if(&a==this) return *this;
17              x = a.x;
18              delete p;
19              p = new int(*a.p);
20              return *this;
21          }
22
23          A& operator =(A&& a)
24          {
25              if(&a == this) return *this;
26              x = a.x;
27              delete p;
28              p = new int(*a.p);
29              a.p = NULL;
30              return *this;
31          }
32
33          A(int m, int n)
34          {
35              p = new int(n);
36              x = m;
37          }
38          ~A()
39          {
40              delete p;
41              x = 0;
42          }
```

```
43
44
45      };
46
47      int main()
48      {
49          A a1(6, 8);
50          A a2;
51          a2 = a1;
52          cout << "a1.x=" << a1.x << ",
53               << "*(a1.p)=" << *(a1.p) << endl;
54          cout << "a2.x=" << a2.x << ",
55               << "*(a2.p)=" << *(a2.p) << endl;
56          cout << "a1.p=" << a1.p << endl;
57          cout << "a2.p=" << a2.p << endl;
58          return 0;
59      }
```

2.2 设计一个日期类 Date

```
1       #include <atltime.h>
2       #include <iostream>
3       class Date
4       {
5       public:
6           friend Date operator+(const Date& date, int day);
7           friend Date operator-(const Date& date, int day);
8           Date()
9           {
10              cur = CTime::GetCurrentTime();
11          }
12          Date(const CTime& date)
13          {
14              cur = date;
15          }
16          Date(int year, int month, int day)
17              :cur(year,month,day,0,0,0)
18          {
19          }
```

```
20
21          Date& operator =(const Date& another)
22          {
23              if (&another == this) return *this;
24              cur = another.cur;
25              return *this;
26          }
27          Date& operator++()
28          {
29              cur += one_day;
30              return *this;
31          }
32          const Date operator++(int)
33          {
34              cur += one_day;
35              return *this;
36          }
37          Date& operator--()
38          {
39              cur -= one_day;
40              return *this;
41          }
42          const Date operator--(int)
43          {
44              cur -= one_day;
45              return *this;
46          }
47          int operator-(const Date& another)
48          {
49              return (cur - another.cur).GetDays();
50          }
51
52          void output() const
53          {
54              std::cout<<cur.GetYear()<<"-"<<cur.GetMonth()<<"-"<<cur.GetDay()<<"\n";
55          }
56      private:
57          CTime cur;
58          static CTimeSpan one_day;
```

```
};
CTimeSpan Date::one_day(1, 0, 0, 0);

Date operator+(const Date& date, int day)
{
    CTimeSpan add_days(day, 0, 0, 0);
    CTime new_cdate(date.cur);
    new_cdate += add_days;
    Date new_date(new_cdate);
    return new_date;
}
Date operator-(const Date& date, int day)
{
    CTimeSpan add_days(day, 0, 0, 0);
    CTime new_cdate(date.cur);
    new_cdate -= add_days;
    Date new_date(new_cdate);
    return new_date;
}
```

2.3 不会越界的 String 类

```
#include<iostream>
#include<cstring>
class String
{

public:
    String()
    {
        str=NULL;
        str_len = 0;
    }
    String(char* s)
    {
        str_len =strlen(s);
        str = new char[str_len+1];
        strcpy(str,s);
    }
```

```
18          char& operator[](int i)
19          {
20              return i>=str_len?str[str_len-1]:str[i];
21          }
22          String& operator=(const String& s)
23          {
24              if(&s==this) return *this;
25              delete[] str;
26              str_len = s.str_len;
27              str = new char[str_len+1];
28              strcpy(str,s.str);
29              return *this;
30          }
31          ~String()
32          {
33              if(!str_len) delete[] str;
34          }
35      private:
36          char* str;
37          int str_len;
38      };
```