类和对象(2)

类的默认成员函数

任何一个类在我们不写的情况下，都会自动生成下面6个默认成员函数

- 初始化和清理
  - 构造函数——完成初始化工作
  - 析构函数——完成清理工作
- 拷贝和复制
  - 拷贝构造——使用同类对象初始化创建对象
  - 赋值重载——把一个对象复制给另一个对象
- 取地址重载
  - 主要是普通对象和const对象取地址

构造函数

构造函数是一个特殊的成员函数，名字与类名相同，创建类类型对象时由编译器自动调用，保证每个数据成员都有一个合适的初始值，并且在对象的生命周期内只调用一次。

特性

1. 函数名与类名相同
2. 无返回值
3. 对象实例化时编译器自动调用对应的构造函数
4. 构造函数可以重载
5. 如果类中没有显式定义构造函数，则编译器会自动生成一个无参的默认构造函数
6. 无参的构造函数和全缺省的构造函数都称为默认构造函数，并且默认构造函数只能有一个
7. 编译器默认生成的无参构造函数，对内置类型的成员变量(int，char)不做任何处理，对自定义类型的成员变量会调用它的构造函数初始化

析构函数

析构函数：对象在销毁时会自动调用析构函数，完成类的一些资源清理工作。(局部对象的销毁工作是编译器完成的)

特性

1. 析构函数名是在类名前加上~
2. 无参数、无返回值
3. 一个类有且只有一个析构函数。若未显示定义，系统会自动生成默认的析构函数
4. 对象生命周期结束时，C++编译器自动调用析构函数
5. 编译器生成的默认析构函数，对内置类型的成员变量(int，char)不做任何处理，对自定义类型成员调用它的析构函数

拷贝构造函数

拷贝构造函数也属于构造函数，只不过是用同类型的对象构造一个新对象 。

特征

1. 拷贝构造函数是构造函数的一个重载形式
2. 拷贝构造函数的形参只有一个且必须使用引用传参(使用传至传参会引发无穷递归调用)
3. 若未显示定义，系统生成默认的拷贝构造函数。默认的拷贝构造函数对象按内存存储按字节序完成拷贝，这种拷贝我们叫做浅拷贝，或者值拷贝。

赋值重载

特征

1. 参数类型
2. 返回值
3. 检测是否自己给自己赋值
4. 返回*this
5. 一个类如果没有显式定义赋值运算符重载，编译器也会生成一个，完成对象按字节序的值拷贝

日期类的实现

```
class Date
{
public:

    //计算每月有多少天
    int GetMonthday(int year, int month)
    {
        //每月0月，所以从下标1开始存储
        int monthday[13] = { 0,31,28,31,30,31,30,31,31,30,31,30,31 };
        //判断闰年，闰年的2月有29天
        if (month == 2 && ((year % 4 == 0 && year % 100 != 0) || year % 400 ==
0))
        {
            return 29;
        }
        return monthday[month];
    }

    //构造函数
    Date(int year = 0, int month = 1, int day = 1)
    {
        if (year >= 0
            && month >= 1 && month <= 12
            && day >= 1 && day <= GetMonthday(year, day))
        {
            _year = year;
            _month = month;
            _day = day;
        }
        else
        {
            cout << "非法日期" << endl;
        }
    }

    //析构函数
```

```
~Date()
{

}

//拷贝构造函数
Date(const Date& d)
{
    _year = d._year;
    _month = d._month;
    _day = d._day;
}

//运算符重载
//d1=d2,赋值运算符
Date& operator=(const Date& d)
    //出了作用域*this还在，则可以使用引用返回，与传值返回相比可以减少一次拷贝构造
{
    //自己不能赋值给自己，d3=d3
    if (this != &d)
    {
        _year = d._year;
        _month = d._month;
        _day = d._day;
    }
    return *this;
}

//比较运算符

//d1==d2
bool operator==(const Date& d)
{
    return _year == d._year && _month == d._month && _day == d._day;
}

//d1!=d2-->!(d1==d2)
bool operator!=(const Date& d)
{
    return !(*this == d);
}

//d1<d2
//bool operator<(Date* this,const Date& d)
bool operator<(const Date& d)
{
    if (_year < d._year)
        return true;
    else if (_year = d._year && _month < d._month)
        return true;
    else if (_year == d._year && _month == d._month && _day == d._day)
        return true;
    return false;
}

//d1<=d2--> d1<d2||d1==d2
//d1.operator<=(&d1,d2)
bool operator<=(const Date& d)
```

```
{
    return *this < d || *this == d;
}

//d1>d2-->!(d1<=d2)
bool operator>(const Date& d)
{
    return !(*this <= d);
}

//d1>=d2-->!(d1<d2)
bool operator>=(const Date& d)
{
    return !(*this < d);
}

//d1+天数
Date operator+(int day)
{
    //拷贝构造一个d1
    Date ret = *this;
    //day是负数
    if (day < 0)
    {
        day = -day;
        return ret - day;
    }
    ret._day += day;
    //加完之后的天数不合法就进入循环
    while (ret._day > GetMonthday(ret._year, ret._month))
    {
        //天数不合法，就往月进位
        ret._day -= GetMonthday(ret._year, ret._month);
        ret._month++;
        //如果月数够了12个月，就往年进位
        if (ret._month == 13)
        {
            ret._year++;
            //月数从1月重新开始
            ret._month = 1;
        }
    }
    return ret;
}

//d1+=10
Date& operator+=(int day)
{
    *this = *this + day;
    return *this;
}

//d1-10
Date operator-(int day)
{
    Date ret = *this;
    //拷贝构造一个d1
    if (day < 0)
```

```
        {
            day = -day;
            return ret + day;
        }
        ret._day -= day;
        //ret._day小于等于0不合法，则去月借天数
        while (ret._day <= 0)
        {
            ret._month--;
            if (ret._month == 0)
            {
                ret._month = 12;
                ret._year--;
            }
            ret._day += GetMonthday(ret._year, ret._month);
        }
        return ret;
    }

    //d1-=10
    Date& operator-=(int day)
    {
        *this = *this - day;
        return *this;
    }

    //后置++，返回加之前的值
    Date operator++(int) //为了构成重载
    {
        //把加之前的值保存起来
        Date tmp(*this);
        *this += 1;
        //返回加之前的值
        return tmp;
    }
    //前置++，返回加之后的值
    Date& operator++()
    {
        *this += 1;
        //返回加之后的值
        return *this;
    }

    //前置--
    Date& operator--()
    {
        *this -= 1;
        return *this;
    }
    //后置--
    Date operator--(int)
    {
        Date tmp(*this);
        *this -= 1;
        return *this;
    }

    //日期相减d1-d2
```

```
    int operator-(const Date& d)
    {
        //符号标志位
        int flag = 1;
        //拷贝构造两个对象
        Date min(*this);
        Date max(d);
        //判断两个日期的大小
        if (min > max)
        {
            min = d;
            max = *this;
            flag = -1;
        }
        //找出小的那个日期，然后进行++，知道两个日期相等
        int day = 0;
        while (min != max)
        {
            //自定义类型，可以调用前置，就不调用后置
            //后置++需要调用拷贝构造
            ++min;
            day++;
        }
        //返回++的次数，就是相差多少天
        return flag * day;
    }

    //打印
    void Print()
    {
        cout << _year << "-" << _month << "-" << _day << endl;
    }
private:
    int _year;
    int _month;
    int _day;
};
```

const成员

将const修饰的类成员函数称之为const成员函数，const修饰类成员函数，实际修饰该成员函数隐含的this指针，表明在该成员函数中不能对类的任何成员进行修改。

取地址及const取地址重载

这两个默认成员函数一般不用重新定义 ， 编译器默认会生成 。