#include <iostream>

**using** **namespace** std;

**class** Date{

**public**:

**int** year,month,day;

    Date(**int** y,**int** m,**int** d):year(y),month(m),day(d){}

    //判断是否为闰年

**bool** isLeapYear(){

**return** (year%100!=0&&year%4==0)||(year%400==0);

    }

    //计算该日在本年中是第几天的成员函数

**int** dayOfyear(){

**int** monthDays[12]={31, 28,31, 30, 31, 30, 31, 31, 30, 31, 30, 31};//各月天数，平年

**int** dayCount=0;

**if**(isLeapYear()){

            monthDays[1]=29;

        }

        //累加前几个月的天数

**for**(**int** i=0;i<month-1;i++){

            dayCount+=monthDays[i];

        }

        //加上当月的天数

        dayCount+=day;

**return** dayCount;

    }

};

**int** main(**int** argc, **const** **char** \* argv[]) {

**int** year,month,day;

    cin>>year>>month>>day;

    // 创建Date对象

    Date date(year, month, day);

    cout<<date.dayOfyear();

**return** 0;

}

在上面的C++程序中，我们设计了一个`Date`类来处理日期相关的操作。这个类主要涉及到两个方面：数据表示和行为。

### 数据表示

- \*\*成员变量\*\*：类中的变量`year`, `month`, `day`用于存储日期的具体信息。它们被定义为`public`，意味着它们可以被类的对象直接访问和修改。

### 行为

- \*\*构造函数\*\*：`Date(int y, int m, int d)`是一个构造函数，它的作用是初始化`Date`类的一个实例。当你创建一个`Date`对象时，你需要提供年、月、日作为参数，这些参数将分别被用来初始化成员变量`year`, `month`, `day`。

- \*\*成员函数\*\*：

- `bool isLeapYear()`：这个函数检查`Date`对象的`year`是否为闰年。闰年的判断规则是：

- 如果年份能被4整除且不能被100整除，那么这一年是闰年。

- 如果年份能被400整除，那么也是闰年。

- 其他情况下，不是闰年。

- 函数返回`true`表示是闰年，`false`表示不是。

- `int dayOfYear()`：这个函数计算`Date`对象所代表的日期在该年中的第几天。它首先根据是否是闰年调整2月的天数（闰年为29天，非闰年为28天），然后累加到该日期前一个月的所有天数，并加上当前月中的天数。这个累加的结果就是该日期在年中的天数。

### 主函数（`main`）

- 程序从标准输入读取年、月、日，然后用这些值创建一个`Date`对象。

- 接着，它调用这个`Date`对象的`dayOfYear()`方法来计算这个日期是年中的第几天，并将结果输出到标准输出。

整体来看，这个程序通过面向对象的方式组织代码，使得日期的处理更为直观和灵活。通过`Date`类的封装，我们可以很容易地创建更多的日期对象，并对每个日期执行相同的操作，比如判断是否闰年或计算年中的天数，而不需要重复代码。