实验 4　类与对象的应用 2

实验目的：

1 掌握类和对象的概念、定义和使用方法。

2 掌握静态数据成员和 const 修饰的成员函数的用法。

3 掌握 c++程序的一般结构。

实验内容：

在实验 3　个人的活期储蓄账户类 SavingsAccount 上修改完成以下内容：

（1）在类 SavingsAccount 中增加一个静态数据成员 total，用来记录各个账户的总金额，并为其增加相应的静态成员函数 getTotal 用来对其进行访问。

（2）将类 SavingsAccount 中不需要改变对象状态的成员函数声明为常成员函数，比如 getBalance 等。

（3）增加日期类 Date

```
class    Date
{
    int    year, month, day;
    int    totalDays;     //该日期是从公元元年 1 月 1 日开始的第几天
public：
    Date(int year, int month, int day);
    int    getYear() const { return year; }
    int    getMonth() const { return month; }
    int    getDay() const { return day; }
    void    show() const;            //输出当前日期
```

bool　isLeapYear() const；//判断当年是否为闰年(计算天数和利息都要用上)

int　distance(const Date& date) const；//计算当前日期与指定日期之间相差天数

};

（4）类 SavingsAccount 中的 int　date 都要改成 Date 类的对象。

（5）将整个程序分为 5 个文件：date.h　account.h 是类定义头文件，date.cpp　account.cpp 是类实现文件，5.cpp 是主函数文件。自己写的头文件引用要用双引号

提示：

（1）利息的计算方式：一年中每天的余额累积起来再除以一年的总天数，得到一个日均余额，再乘以年利率。

（2）两个日期相差天数的计算方式：选取一个基准日期（如公元元年 1 月 1 日），在计算两个日期相差天数时，先分别将两个日期与基准日期的相对天数计算出来，再将两个相对天数相减即可。

（3）与基准日期（如公元元年 1 月 1 日）相对天数的计算方式：（1）计算公元元年到公元 y-1 年的总天数。平均每年有 365 天，闰年多一天，即 365*（y-1）加上公元元年到 y-1 年之间的闰年数。（2）加上当年当月 1 日到当年 1 月 1 日之间相差的天数。（3）加上当年当月当日到当年当月 1 日之间相差的天数。

（4）可以把每月 1 日到 1 月 1 日天数放在一个数组中，该数组元素值分别是：0，31，59，90，120，151，181，212，243，73，304，334，365

（5）两个头文件里先写：

#ifndef _DATE_H_	#ifndef _ACCOUNT_H_
#define _DATE_H_	#define _ACCOUNT_H_
…	…
#endif	#endif

要求：

- 完成上述成员函数的定义；
- 定义类对象，测试程序的正确性

定义两个账户 s0 和 s1，年利率都是 1.5%，都在 2020 年 11 月 1 日创建账户，随后 s0 在 2020 年 11 月 5 日和 2020 年 12 月 5 日分别存入 5000 元和 5500 元，s1 在 2020 年 11 月 25 日和 2020 年 12 月 20 日分别存入 10000 元和取出 4000 元。2021 年 1 月 1 日是银行的计息日。分别输出 s0(17 的样子)和 s1(12 的样子)两个账户的信息(账号、余额)及所有账户的总额。

⑶写出实验报告。

➢ Date.cpp

```
#include "Date.h"

using namespace std;

int Date::SumOfMonth[12] = {0, 31, 59, 90, 120, 151, 181, 212, 243, 273, 304, 334};

Date::Date(int year, int month, int day) : year(year), month(month), day(day), totalDays(0) {};

void Date::show() const
{
    cout << year << "年" << month << "月" << day << "日";
}

bool Date::isLeapYear() const
```

```
{
    return (year % 4 == 0 && year % 100 != 0) || (year % 400 == 0);
}

int Date::distance(const Date &date) const
{
    // 计算从基准日期（第1年）到当前日期的天数
    int current_days = (year - 1) * 365;
    for (int i = 1; i < year; i++)
    {
        if ((i % 4 == 0 && i % 100 != 0) || (i % 400 == 0))
            current_days++;
    }
    current_days += SumOfMonth[month - 1] + day;
    if (isLeapYear() && month > 2)
        current_days++;

    // 计算从基准日期（第1年）到参数日期的天数
    int param_days = (date.year - 1) * 365;
    for (int i = 1; i < date.year; i++)
    {
        if ((i % 4 == 0 && i % 100 != 0) || (i % 400 == 0))
            param_days++;
    }
    param_days += SumOfMonth[date.month - 1] + date.day;
    if (date.isLeapYear() && date.month > 2)
        param_days++;

    // 返回两个日期之间的差值
    return param_days - current_days;
}
```

- Date.h

```
#ifndef DATE_H
#define DATE_H

#include <iostream>

class Date
{
private:
    int year, month, day;
    int totalDays;              // 该日期是从公元元年1月1日开始的第几天
    static int SumOfMonth[12]; // 每月1日到1月1日的天数
```

```
public:
    Date(int year, int month, int day);
    inline int getYear() const { return year; }
    inline int getMonth() const { return month; }
    inline int getDay() const { return day; }
    void show() const;                    // 输出当前日期
    bool isLeapYear() const;              // 判断当年是否为闰年
    int distance(const Date &date) const; // 计算当前日期与指定日期之间相差天数
};

#endif
```

➢ SavingsAccount.cpp

```
#include "SavingsAccount.h"

using namespace std;

double SavingsAccount::total = 0;

SavingsAccount::SavingsAccount(int _id, const Date &_date, double _rate)
    : id(_id), balance(0), rate(_rate), lastDate(_date), accumulation(0) {};

double SavingsAccount::accumulate(const Date &date)
{
    int days = date.distance(lastDate);
    // 确保天数是正的
    days = abs(days);
    accumulation += balance * days;
    lastDate = date;
    return accumulation;
}

void SavingsAccount::deposit(const Date &date, double amount)
{
    accumulate(date);
    balance += amount;
    total += amount;
    date.show();
    cout << " 已存入" << amount << "元" << endl;
    cout << "当前余额为" << balance << "元" << endl;
}

void SavingsAccount::withdraw(const Date &date, double amount)
{
```

```
    accumulate(date);
    balance -= amount;
    total -= amount;
    date.show();
    cout << " 已取出" << amount << "元" << endl;
    cout << "当前余额为" << balance << "元" << endl;
}

void SavingsAccount::settle(const Date &date)
{
    // 计算日均余额
    double dailyBalance = accumulate(date) / (date.isLeapYear() ? 366 : 365);
    // 计算利息
    double interest = dailyBalance * rate;
    balance += interest;
    total += interest;
    accumulation = 0;
    cout << "当前利息为" << interest << "元" << endl;
}

void SavingsAccount::show() const
{
    cout << "账号:" << id << endl;
    cout << "当前余额为:" << balance << "元" << endl;
}
```

- SavingsAccount.h

```
#ifndef ACCOUNT_H
#define ACCOUNT_H

#include "Date.h"

class SavingsAccount
{
private:
    int id;             // 帐号
    double balance;     // 余额
    double rate;        // 年利率
    Date lastDate;      // 上次变更余额的日期
    double accumulation; // 余额按日累加之和
    static double total; // 所有账户的总金额
public:
    SavingsAccount(int id, const Date &date, double rate); // 构造函数
```

```
    double accumulate(const Date &date);                    // 获得到指定日期为止的存款金额按日累积
值
    void deposit(const Date &date, double amount);          // 存入现金，date 为日期，amount 为金额
    void withdraw(const Date &date, double amount);         // 取出现金
    void settle(const Date &date);                          // 结算利息，每年 1 月 1 日调用一次该函数
    void show() const;                                      // 输出账户信息
    inline int getId() const { return id; }
    inline double getBalance() const { return balance; }
    inline double getRate() const { return rate; }
    static inline double getTotal() { return total; } // 获取所有账户总金额
};

#endif
```

➢ Main.cpp

```
#include "SavingsAccount.h"
#include <iostream>

using namespace std;

const double rate = 0.015;

int main()
{
    Date date1(2020, 11, 1);  // 创建账户日期
    Date date2(2020, 11, 5);  // s0 第一次存款日期
    Date date3(2020, 12, 5);  // s0 第二次存款日期
    Date date4(2020, 11, 25); // s1 第一次存款日期
    Date date5(2020, 12, 20); // s1 取款日期
    Date date6(2021, 1, 1);   // 计息日期

    SavingsAccount s0(1, date1, rate);
    SavingsAccount s1(2, date1, rate);

    s0.deposit(date2, 5000);
    s0.deposit(date3, 5500);
    s1.deposit(date4, 10000);
    s1.withdraw(date5, 4000);

    s0.settle(date6);
    s1.settle(date6);

    s0.show();
    s1.show();
```

```
    cout << "所有账户总金额为: " << SavingsAccount::getTotal() << "元" << endl;

    return 0;
}
```

➢ 运行结果

```
(pt2) PS D:\code\Experimental_Report\CPP\面向对象基本知识\E3_类与对象的应用2\c1>    g++ Main.cpp
ate.cpp SavingsAccount.cpp -o Main
(pt2) PS D:\code\Experimental_Report\CPP\面向对象基本知识\E3_类与对象的应用2\c1> Main
2020年11月5日 已存入5000元
当前余额为5000元
2020年12月5日 已存入5500元
当前余额为10500元
2020年11月25日 已存入10000元
当前余额为10000元
2020年12月20日 已取出4000元
当前余额为6000元
当前利息为17.8151元
当前利息为13.2329元
账号:1
当前余额为:10517.8元
账号:2
当前余额为:6013.23元
所有账户总金额为: 16531元
```

三、 实验总结

- **静态数据成员和常成员函数的应用:**

静态成员可以共享类的状态，避免每个对象单独存储相同的数据；常成员函数能保证不修改对象状态。在实践中，我应用了静态数据成员 total 来跟踪所有账户的总金额，确保每次账户的变动都能更新这一共享值。同时，使用 const 修饰成员函数，如 getBalance，确保这些函数不会修改类的状态。

- **日期处理：**

通过实现 Date 类，我掌握了日期的内部表示和计算方法，特别是如何计算日期差和判断闰年。相比传统的使用 int 类型表示日期，使用 Date 类能够更精确地处理日期相关操作。

- **程序结构与模块化：**

将程序分为多个文件进行管理，增强了代码的可读性和可维护性。我也更加熟悉了 CPP 中各个文件的引用及其编写规范。

- **内联函数的使用：**

Date 和 SavingsAccount 类中有对应的 get 方法,我尝试将其改写为了课堂中学习到的内联函数,能够提高程序的运行效率，但内联函数会将函数体复制到每个调用处，这可能导致程序的整体体积增加。