

实验十 对象与类的进一步讨论（特殊的成员）

一、 问题描述

1. 实验目的：

掌握const、static、友元的若干基本概念和特性，并能够应用于程序编写。

掌握验证性实验的基本方法和过程(认知、实验、总结)。

2. 实验内容：

分别编写一段测试代码来回答任务书中的相关问题（每一个问题，用一个工程文件，同时需要记录相应的调试过程），具体问题请参考“实验任务说明10.doc”；

调试的过程；（动态调试的相关截图，比如 设置断点、查看当前变量值等）；

编译出来的可执行程序单独放在一个目录下（bin/exe/debug目录下，同时附上输入数据说明和输出结果）

二、 实验过程

1. 关于const专题

（一）改错题

1、对于常成员函数，下面描述正确的是（ C ）

//备注：用标注的形式给出对每个选项的分析

A 常成员函数只能修改常数据成员

//常对象必须初始化，并且初始化后不能修改数据成员，错误。

B 常成员函数只能修改一般数据成员

//常对象必须初始化，并且初始化后不能修改数据成员，错误。

C常成员函数不能修改任何数据成员

//常对象必须初始化，并且初始化后不能修改数据成员，正确。

D 常成员函数只能通过常对象调用

//非常对象也能调用常成员函数，若常成员函数是重载的,只有常对象才能调用。

2、分析并改错，分析错误原因

```
#include <iostream>
using namespace std;
class Init {
private:
    const int num;
    int count;
public:
    Init() {
        num = 1;
        count = 10;
    }
    void print() { cout << num << count << endl; }
};
void main() {
    Init obj;
    obj.print();
}
```

答：这里num是常数据成员，故无法进行修改，可以在构造函数初始化列表显示赋值。

```
#include <iostream>
using namespace std;
class Init {
private:
    const int num;
    int count;
public:
    Init():num(1) {
        count = 10;
    }
    void print() { cout << num << count << endl; }
};
void main() {
    Init obj;
    obj.print();
}
```

（二）代码分析题

1、分析代码中出现的问题（每个问题给出界面的截图，并进行分析）

本题主要关注：常引用（经常做形式参数，分别考虑const和&的作用）

```

#include <iostream>
using namespace std;
int add(const int& i, const int& j);
int main() {
    int a = 20;
    int b = 30;
    cout << a << " + " << b << " = " << add(a, b) << endl;
    return 0;
}
int add(const int& i, const int& j) {
    i = i + 30;
    j = j - 10;
    return i + j;
}

```

答：这里定义了常引用*i,j*，不能通过该引用修改它所引用的变量（对象）。一般用常引用做形参，在函数中不能更新引用所绑定的对象，便不会意外地发生对实参的更改（即：只能访问）修改代码如下：

```

#include <iostream>
using namespace std;
int add(const int& i, const int& j);
int main() {
    int a = 20;
    int b = 30;
    cout << a << " + " << b << " = " << add(a, b) << endl;
    return 0;
}
int add(const int& i, const int& j) {
    int k = i + 30;
    int r = j - 10;
    return k + r;
}

```

2、完善代码

本题主要关注：常数据成员

关于常数据成员，先思考：为何要有常数据成员，再考虑成员初始化列表

```

#include <iostream>
using namespace std;
class Date {
    Date(int y,int m,int d);
    void showDate();
}

```

```

private:
    const int year;
    const int month;
    const int day;
};
Date::Date(int y,int m,int d):year(y),month(m),day(d) { } //函数体中为空
void Date::showDate() {
    cout<<year<<"."<<month<<"."<<day<<endl; }
int main() {
    Date date1(2009,10,15);
    date1.showDate(); //非const成员函数可以访问const的数据成员
    return 0;
}
class Date {
public:
    Date(int y, int m, int d);
    void showDate();
private:
    const int year;
    const int month;
    const int day;
};

Date::Date(int y, int m, int d):year(y),month(m),day(d) { } //函数体中为空

void Date::showDate() {
    cout << year << "." << month << "." << day << endl;
}
int main() {
    Date date1(2019, 10, 25);
    void showDate();
    return 0;
}

```

(1) 请回答，在类的声明过程中，什么时候会描述const的数据成员？

答：常成员可以提供共享数据的保护。当希望保护共享数据时，可以使用const去描述数据成员。

(2) 若 cout<<year<<"."<<month<<"."<<day<<endl; 语句修改为
year=year+1; month=month+1; day=day+1; 会有什么结果？

答：将出现表达式必须是可修改的左值的报错。year, month, day是常数据成员。常数据成员必须进行初始化，并且不能被更新。常数据成员的初始化只能通过构造函数的成员初始化列表显式进行。

```

class Date {
public:
    Date(int y, int m, int d);
    void showDate();
private:
    const int year;
    const int month;
    const int day;
};

Date::Date(int y, int m, int d):year(y),month(m),day(d) { } //函数体中为空

void Date::showDate() {
    cout << year << "." << month << "." << day << endl;
    year = year + 1; month = month + 1; day = day + 1;
}

int main() {
    Date date1(2019, 10, 25);
    void showDate();
    return 0;
}

```

3、测试下列代码（即，针对代码，修改进行，验证性测试。给出界面截图）

（1）关注：常成员函数的特性

特性分析：声明和定义都需要写const，且可以作为重载的识别信息；

<pre> class Sample { int n; public: Sample(int i){ n=i;}; void print() const; }; void main() { const Sample a(10); a.print(); } void Sample::print() const { cout<<"2:n="<<n<<endl; } </pre>	<pre> #include <iostream.h> class Sample { int n; public: Sample(int i){ n=i;}; void print() { cout<<"1:n="<<n<<endl;} void print() const{ cout<<"2:n="<<n<<endl;} }; void main() { const Sample a(10); Sample b(20); a.print(); b.pirnt(); } </pre>
--	--

```

#include <iostream>
using namespace std;

class Sample
{
    int n;
public:
    Sample(int i) { n = i; };
    void print() const;
};

void main() {
    const Sample a(10);
    a.print();
}

void Sample::print() const {
    cout << "2:n=" << n << endl;
}

```

```

#include <iostream>
using namespace std;

class Sample
{
    int n;
public:
    Sample(int i) { n = i; };
    void print() { cout << "1:n=" << n << endl; }
    void print() const {
        cout << "2:n=" << n << endl;
    }
};

void main() {
    const Sample a(10);
    Sample b(20);
    a.print();
    b.print();
}

```

答：常成员函数声明的格式：<返回类型> <成员函数名>(参数列表) const。const 是函数类型的一个组成部分，因此在函数声明和实现部分中都要带有 const 关键字；若定义常成员函数时丢失了 const 关键字，程序会产生错误。

特性分析：const 成员函数中语句不能修改数据成员，且也只能调用 const 成员函数；

```

#include <iostream.h>
class Sample
{
    int n;
public:
    Sample(int i){ n=i;};
    void print() const;
    void setValue(){ n =100;}
};
void main()
{
    const Sample a(10);
    a.print();
}
void Sample::print() const
{
    n=200;           //error C2166: l-value specifies const object
    setValue();       //error C2662: 'setValue' : cannot convert 'this' pointer
                      //from 'const class Sample' to 'class Sample &'

    cout<<"2:n="<<n<<endl;
}
#include <iostream>
using namespace std;

class Sample {
    int n;
public:
    Sample(int i) { n = i; };
    void print()const;
    void setValue() { n = 100; }
};

void main() {
    const Sample a(10);
    a.print();
}
void Sample::print()const
{
    //n = 200;
    //error C2166:l-value specifies const object
    //setValue();
    //error C2662:"setValue"cannot convert this pointer
    // from const class Sample to class Sample &
    cout << "2:n=" << n << endl;
}

```

答：常成员函数不能修改更新对象的数据成员。常成员函数只能调用const的常成员函数。应用场景：用于读取数据成员或是可能出现常对象的情况。

（2）关注常对象与成员函数间关系

特性分析：常对象只能调用const成员函数


```

#include <iostream.h>
class Sample
{
    int n;
public:
    Sample(int i){ n=i;};
    void print() { cout<<"1:n="<<n<<endl;};
};

void main()
{
    const Sample a(10);
    a.print();           //error C2662: 'print' : cannot convert 'this' pointer
                        //from 'const class Sample' to 'class Sample &'
                        //Conversion loses qualifiers
}
}

#include <iostream>
using namespace std;
class Sample
{
    int n;
public:
    Sample(int i) { n = i; };
    void print() { cout << "1:n=" << n << endl; }
};

void main() {
    const Sample a(10);
    //a.print();
    //error:print cannot convert this pointer
    //from const class Sample to class Sample
    //conversion lossier qualifiers
}
}

```

答：const关键字可以用于参与对 重载函数的区分。原则是：常 对象调用常成员函数，一般对 象调用一般成员函数。除此之外，常对象只能调用const成员函数，所以这里a无法调用不是const成员函数的print()。

二、关于友元专题

（一）选择与简答题

1、什么是友元机制？友元有哪些分类？

答：友元机制是类外的一般函数或是另一个类的成员函数能够对该类体中的私有成员和其他成员能直接访问；友元是以牺牲信息隐藏原则，削弱封装性来提高编程 效率，增大了类与类之间的耦合度；友元的种类：（1）一般函数：类外的函数（2）另一个类的成员函数（3）另一个类：其他类全部的成员函数。特点：都是在本类外，友元不是被访问类的成员，所以它不受类的访问权限影响

2、下面友员的错误描述是（D）【对错误项进行描述分析】

A 关键字friend用于声明友员

B 一个类中的成员函数可以是另一个类的友员

C 友员函数访问对象的成员不受访问特性影响

D 友员函数通过this指针访问对象成员

// this指针是在指向类成员本身，但是友元函数并不在类里面，而是在类外面。

（二）程序填空题

1、分析并填充程序

```
#include <iostream>
```

```
using namespace std;
```

```
class Time
```

```
{ public:
```

```
    Time(int,int,int);
```

```
    friend void display(const Time& t);//友元函数的声明
```

```
    void prin();
```

```
private:
```

```
    int hour;
```

```
    int minute;
```

```
    int sec;
```

```
};
```

```
Time::Time(int h,int m,int s)
```

```
{ hour=h;
```

```
  minute=m;
```

```
  sec=s;
```

```
}
```

```
Time::prin()    { cout<<hour<<minute<<sec; }
```

```
void display(const Time &t)
```

```
{ cout<<t.hour<<":"<<t.minute<<":"<<t.sec<<endl;}    //直接输出对象的hour、minute和sec
```

```
int main()
```

```
{ Time t1(10,13,56);
```

```
  display(t1);
```

```
  t1.prin(); //成员函数的调用
```

```
  return 0;
```

```
}
```

（1）请分析：display()和prin()函数的区别

答：print()函数是类Time的成员函数，可以被Time类型的对象t1调用。display在Time类中声明了友元函数，最后也成功访问到了t1对象的保护成员。友元函数并不能看做是类的成员函数，它只是个被声明为类友元的普通函数。

(2) 请分析：display()函数的参数和函数体的书写特点

答：friend void display(const Time& t)。友元函数说明格式： friend <类型> <函数名>(<参数列表>) 特性：可以访问与其有好友关系的类中的私有成员。因为友元函数没有this指针，则参数要有三种情况：（1）要访问非static成员时，需要对对象做参数；（2）要访问static成员或全局变量时，则不需要对象做参数；（3）如果做参数的对象是全局对象，则不需要对象做参数；

```
#include <iostream>
using namespace std;

class Time{
public:
    Time(int, int, int);
    friend void display(const Time& t); //友元函数的声明
    void prin();
private:
    int hour;
    int minute;
    int sec;
};
//构造函数
Time::Time(int h, int m, int s)
{
    hour = h;
    minute = m;
    sec = s;
}

void Time::prin() { cout << hour << minute << sec; }

void display(const Time &t)
{
    cout << t.hour << ":" << t.minute << ":" << t.sec << endl;
    //直接输出对象的hour、minute和sec
}

int main()
{
    Time t1(10, 13, 56);
    display(t1);
    t1.prin(); //成员函数的调用
    return 0;
}
```

2、在没有描述运算符重载之前，实现相关操作，比如加的操作，是通过定义成员函数或友元来实现。请完善代码

```
#include <iostream>
using namespace std;

class Complex
{
private:
    double real;
    double im;
public:
    Complex(double r, double i) {
        real = r;
        im = i;
    }
    Complex(Complex& t) {
        real = t.real;
        im = t.im;
        cout << "HAHA    ";
    }
    Complex() {}
    Complex& Add(const Complex& t1) //要求完善代码，实现复数的加
    {
        Complex newnum;
        newnum.real = t1.real + real;
        newnum.im = t1.im + im;
        return newnum;
    }
    friend Complex& Add2(const Complex& t1, const Complex& t2);
    void display() { cout << real << ":" << im << endl; }
};
```

```

Complex& Add2(const Complex& t1, const Complex& t2) //要求完善代码，实现复数的加
{
    Complex newnum;
    newnum.real = t1.real + t2.real;
    newnum.im = t1.im + t2.im;
    return newnum;
}

int main() {
    Complex p1(11, 12), p2(13, 14);
    Complex c1, c2;
    c1 = p1.Add(p2);
    c2 = Add2(p1, p2);
    c1.display();
    c2.display();
    return 1;
}

```

(1) 根据add和add2位置特性，完善代码；

答：如上图所示。

(2) 在main中添加语句：c2=p1+p2; 程序的执行情况反馈

答：出现报错，没有与这些操作符匹配的+运算符。

(3) 对main函数中下列语句，重点分析执行过程

c1=p1.Add(p2);c2=Add2(p1,p2);

答：Add()函数是类Complex的成员函数，可以被Complex类型的对象调用。Add2在Complex类中声明了友元函数，最后也成功访问到了对象的保护成员。友元函数并不能看做是类的成员函数，它只是个被声明为类友元的普通函数。

(4) 结合(2)和(3)可以看出，上述代码在功能上实现了两个复数p1和p2的“加”，但是无法满足用户对“p1+p2”表述书写的需求，该如何做？

答：可以对运算符重载。

(5) 请描述程序中复制构造函数调用了几次？分别是在什么时候调用的？

答：代码中没有调用复制构造函数。

 选择 Microsoft Visual Studio 调试控制台

```

调用两参数构造函数
调用两参数构造函数
调用无参构造函数
调用无参构造函数
调用无参构造函数
调用无参构造函数
24:26
24:26

```

（三）改错题

要求：

- （1）对错误的代码做修改并标注
- （2）取消程序中的友元，采用普通函数完成函数功能【通过增加访问类中保护数据的const成员函数】

① 使用友元

```
#include <iostream>
class Animal;
void setValue(Animal&, int);
void setValue(Animal&, int, int);

class Animal {
public:
    friend void SetValue(Animal&, int);
    //需要声明友元
    friend void SetValue(Animal & ta, int tw, int tn);
protected:
    int itsWeight;
    int itsAge;
};

void SetValue(Animal& ta, int tw) {
    ta.itsWeight = tw;
}

void SetValue(Animal& ta, int tw, int tn)
{
    ta.itsWeight = tw;
    ta.itsAge = tn;
}

int main() {
    Animal peppy;
    SetValue(peppy, 5);
    SetValue(peppy, 7, 9);
    return 0;
}
```

② 取消友元

```

#include <iostream>
class Animal;
void SetValue(Animal&, int);
void SetValue(Animal&, int, int);

class Animal {
public:
    void SetValue(Animal&, int);
    //需要声明友元
    void SetValue(Animal& ta, int tw, int tn);
protected:
    int itsWeight;
    int itsAge;
};

void Animal::SetValue(Animal& ta, int tw) {
    ta.itsWeight = tw;
}

void Animal::SetValue(Animal& ta, int tw, int tn)
{
    ta.itsWeight = tw;
    ta.itsAge = tn;
}

int main() {
    Animal peppy;
    SetValue(peppy, 5);
    SetValue(peppy, 7, 9);
    return 0;
}

```

（四）代码题：计算二维坐标系中，两点之间的距离。

(1) 完善类的声明

答：如下图。

(2) 在类外定义计算两点之间距离的成员函数、计算两点之间距离的普通函数

答：如下图。

(3) 完善主函数，比较Distance和NewDistance的应用上的区别

答：如下图。输出结果相同。NewDistance()函数是类Point的成员函数，可以被Point类型的对象调用。Distance在Point类中声明了友元函数，最后也成功访问到了对象的保护成员。友元函数并不能看做是类的成员函数，它只是个被声明为类友元的普通函数。

```

#include <iostream>
using namespace std;

class Point {
    double x, y;
public:
    Point(double xx, double yy): x(xx), y(yy) {} ;//参数初始化列表
    //计算两点距离的普通函数
    friend double Distance(const Point& p1, const Point& p2);
    double NewDistance(const Point& p1) ; //计算两点距离的成员函数
};

//计算两点之间距离的成员函数
double Point::NewDistance(const Point& p1) {
    double dis = sqrt(pow((p1.x - x), 2) + pow((p1.y - y), 2));
    return dis;
}

//计算两点距离的普通函数
double Distance(const Point&p1, const Point& p2) {
    double dis = sqrt(pow((p1.x - p2.x), 2) + pow((p1.y - p2.y), 2));
    return dis;
};

int main() {
    Point a(2, 3); Point b(1, 2);
    cout<<a.NewDistance(b)<<endl;
    cout << Distance(a,b) << endl;
}

```

Microsoft Visual Studio 调试控制台

```

1. 41421
1. 41421

```

(4) 思考：由于将Distance描述成友元，那么它可以访问对象的私有成员，实现两点间距离的计算。现在若要求在不引入友元前提下，仍实现两点之间距离的计算，该如何实现？

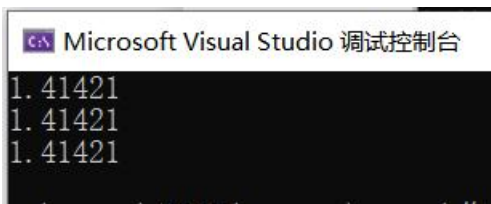
答：可增加public的成员函数，成员函数来访问私有成员；普通函数访问public的成员函数

```

//不使用friend的普通函数
double MyDistance(Point& p1, Point& p2) {
    double dis = sqrt(pow((p1.Getx() - p2.Getx()), 2) + pow((p1.Gety() - p2.Gety()), 2));
    return dis;
};

int main() {
    Point a(2, 3); Point b(1, 2);
    cout<<a.NewDistance(b)<<endl;
    cout << Distance(a,b) << endl;
    cout << MyDistance(a, b) << endl;
}

```

```
Microsoft Visual Studio 调试控制台
1. 41421
1. 41421
1. 41421
```

三、静态成员专题

(一) 简答与选择题

1、静态成员的分类？

答：C++提供static关键字来声明静态成员，生成“类属性”，即该属性为整个类所共有，不属于任何一个具体对象。静态成员在每个类中只有一个副本，由该类的所有对象共同维护和使用；使用静态成员，来解决同一个类的不同对象之间的数据与函数共享问题，是类的所有对象共享的成员，而不是某个对象的成员，分为：静态数据成员，静态成员函数

2、静态数据成员有什么特性？如何声明、定义和初始化、使用？

答：（1）静态数据成员是一种特殊的数据成员，它以关键字static开头；语句格式：class 类名 { ...static 类型说明符 成员名; ... };定义static数据成员，能够实现类的所有对象 在类的范围内共享某个数据。

（2）类的静态数据成员视为该类类型的全局变量。静态数据成员在每个类对象中并不占有存储空间，是对每个类分配有存储空间；静态数据成员只有一份，它为各对象所共有【不是独立拥有】，并不只属于某个对象，所有对象都可以引用它，实现同类中不同对象之间的数据共享；在定义对象之前，就有静态数据成员；对于所有对象而言，静态数据成员的值是一样的；对于非静态数据成员，每个类对象中都有自己的副本；

（3）在类的声明中仅是对静态数据成员进行声明，还需要对静态数据成员进行定义和初始化。形式：数据类型 类名::静态数据成员名=初值;初始化不能在类声明或是类的构造函数中进行【相对对象，静态数据成员是独立的】，只能在类体外进行。此时不再写static关键字，需要使用域运算符::。若未对静态数据成员赋初值,则自动赋予初值（根据类型不同，值也不同）。静态数据成员是在所有对象之外单独开辟空间。只要在类中定义了静态数据成员，即使不定义对象，也为静态数据成员分配空间，可以被引用。（4）

（4）在类的public部分说明的静态数据成员 通过对象名或是类名，即： 对象名.静态数据成员 或 类名::静态数据成员 这可传达：静态数据成员并不是属于或依赖某个对象,而是属于类的，但同类的对象可以共享引用它。在类的非public部分说明的静态数据成员 只能由类的成员函数访问。

3、静态成员函数有什么特性？如何声明、定义和调用？

答：（1）静态成员函数定义。在类中声明函数的前面加static就成了静态成员函数。语句格式：class 类名 { ... static 类型 函数名(形参) { 函数体 } ... }; 若是在类外定义static成员函数，则无须重复指定static关键字；static成员函数不能

被声明为const;

(2) static成员函数特性: 是类的一部分, 而不是对象的一部分。它可以直接访问所属类的static成员, 但由于没有this形参, 不能直接使用非static成员 (原因: 由于调用时可不捆绑具体对象, 被调用时, 无当前对象信息。若要访问非静态成员时【访问对象的私有空间】, 必须通过参数传递的方式得到相应的对象, 再通过对象来访问。)

(3) 静态成员函数的使用: 在类外调用静态成员函数 类名::静态成员函数 或对象名.静态成员函数。

4、下面对静态数据成员的描述中, 正确的选项 (D) 【对错误的选项进行分析】

A 静态数据成员可以在类体内进行初始化

//只有基本类型的静态常量才可以在类内初始化。

B 静态数据成员不可以被类的对象调用

//静态数据成员可以被类的对象调用。

C 静态数据成员不受private控制符作用

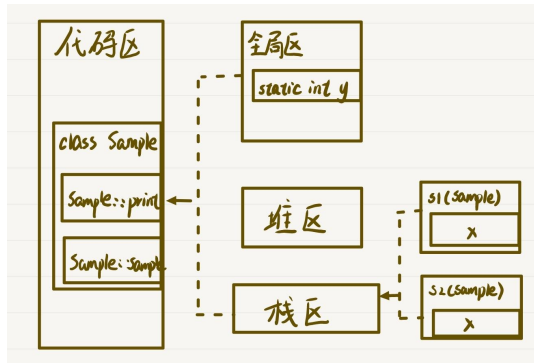
//静态数据成员可以被private之类的修饰作用, 被private修饰之后, 只有类的友元和成员函数可以访问。

D 静态数据成员可以直接用类名调用

(二) 程序阅读题: 分析程序的运行结果, 思考静态成员的使用

<p>1、画出y空间的结构示意图, 分析结果</p> <pre>#include <iostream> class Sample { private: int x; static int y; //语句1 public: Sample(int a) { x=a; x++; y++; } void print() { cout<<"x="<<x<<"",y="<<y<<endl; } }; int Sample::y=10; void main() { Sample s1(20); Sample s2(30); s1.print(); s2.print(); }</pre> <p>答: 静态数据成员并不是属于或依赖某个对象,而是属于类的, 但同类的对象可以共</p>	<p>1、分析static成员函数, 找出错误分析原因;</p> <pre>#include <iostream> class Sample { private: int x; static int y; public: Sample(int a) { x=a; y+=x; } static void print(Sample s) { cout<<"x="<<x<<"",y="<<y<<endl; } }; int Sample::y=10; void main() { Sample s1(20); Sample s2(30); s1.print(s1); s2.print(s2); }</pre> <p>答: 非静态成员引用必须与对象相对。这里将x改为s.x成功运行。</p>
--	--

享引用它。在类的非public部分说明的静态数据成员 只能由类的成员函数访问。



2、在main中，直接添加： `cout<<Sample::y;` 请问程序的运行结果如何？并解释原因。

答：出现成员不可访问的报错，在类的非public部分说明的静态数据成员 只能由类的成员函数访问。

3、若将语句1调整至public:下，main函数中只保留并执行这语句： `cout<<Sample::y;` 请问程序的运行结果如何？并解释原因。

答：报错消失，程序成功运行。类的public部分说明的静态数据成员 可以由类的成员函数外的函数访问。

2、用文字总结static成员函数特性

答：（1）在类中声明函数的前面加static就成了静态成员函数。若是在类外定义static成员函数，则无须重复指定static关键字；static成员函数不能被声明为const；

（2）static成员函数是类的一部分，而不是对象的一部分。它可以直接访问所属类的static成员，但由于没有this形参，不能直接使用非static成员（原因：由于调用时可不捆绑具体对象，被调用时，无当前对象信息。若要访问非静态成员时【访问对象的私有空间】，必须通过参数传递的方式得到相应的对象，再通过对象来访问。）。

（3）静态成员函数的使用：在类外调用静态成员函数 类名::静态成员函数 或 对象名.静态成员函数。

3、代码题

某公司由若干个子组成，Budget类用来计算公司的预算。该类包含一个静态的数据成员CorpBudget，用来存储整个公司的预算额。当调用函数成员addBudget时，将参数增加到CorpBudget中。程序结束时，CorpBudeget的值将是整个公司的预算额。

根据描述，尽可能完善代码。

思路：每个子公司都是一个对象。static的数据成员往往用于统计功能。

```

#pragma once
// Budget 类的定义
class Budget
{
private:
    static float CorpBudget; // 存储整个公司的预算额
    float divBudget;
public:
    // 构造函数
    Budget() { divBudget = 0; }
    void addBudget(float b)
    {
        divBudget += b;
        CorpBudget += divBudget; // 引用静态数据成员
    }
    float getDivBudget() { return divBudget; }
    float getCorpBudget() { return CorpBudget; } // 引用静态数据成员
};

#include <iostream>
using namespace std;
#include "budget.h"
float Budget::CorpBudget = 0;

int main()
{
    int count, subnum;
    float bud;
    cout << "输入子公司数量: ";
    cin >> subnum;
    Budget *divisions = new Budget[subnum];
    for (count = 0; count < subnum; count++)
    {
        cout << "输入子公司 " << (count + 1) << " 预算额: ";
        cin >> bud;
        divisions[count].addBudget(bud);
    }

    cout.precision(2);
    cout.setf(ios::showpoint | ios::fixed);

    cout << "\n公司预算如下:\n";
    for (count = 0; count < subnum; count++)
    {
        cout << "\t 子公司 " << (count + 1) << " 预算 ";
        cout << divisions[count].getDivBudget() << endl;
    }

    cout << "\t 公司总预算 ";
    cout << divisions[0].getCorpBudget() << endl;
    return 0;
}

```

```
Microsoft Visual Studio 调试控制台
输入子公司数量: 3
输入子公司 1 预算额: 111
输入子公司 2 预算额: 222
输入子公司 3 预算额: 333

公司预算如下:
    子公司 1 预算 111.00
    子公司 2 预算 222.00
    子公司 3 预算 333.00
    公司总预算 666.00
```

三、 附录

源程序文件项目清单: src/代码题文件夹 src/其他题文件夹