**C++第十次实验报告 对象与类的进一步讨论（特殊的成员）**

**黄勖 22920212204392**

一、**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 ）

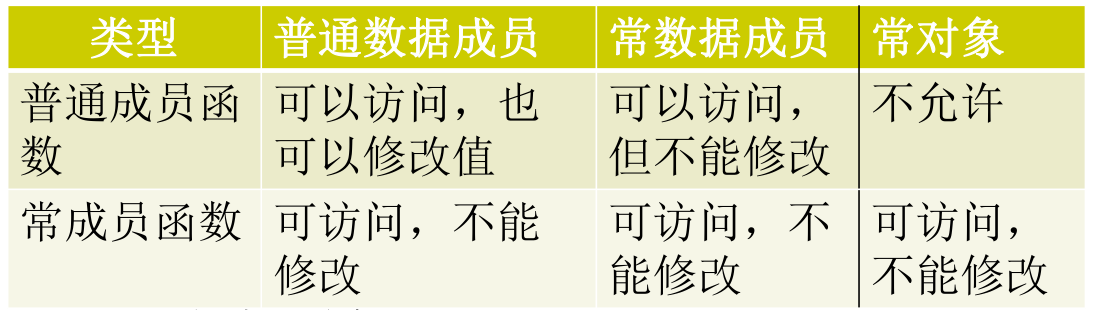
**//备注：用标注的形式给出对每个选项的分析**

A 常成员函数只能修改常数据成员 常成员函数都不能修改

B 常成员函数只能修改一般数据成员 常成员函数都不能修改

C常成员函数不能修改任何数据成员

D 常成员函数只能通过常对象调用 普通对象也可以调用常成员函数



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; }

};

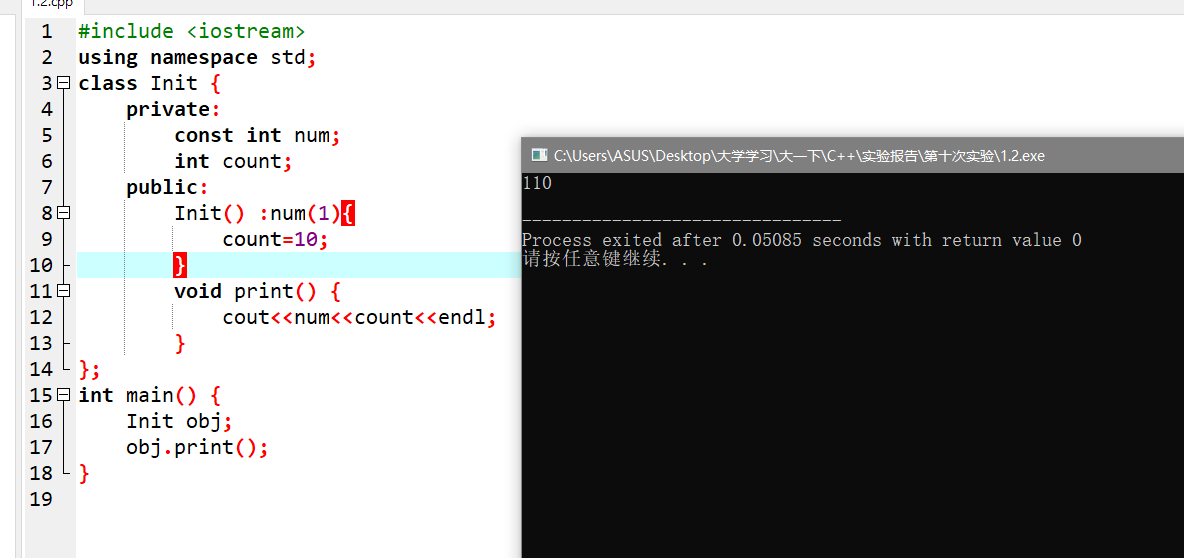
void main() {

Init obj;

obj.print();

}

**原因：常数据成员应使用列表初始化！**



**（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;

}

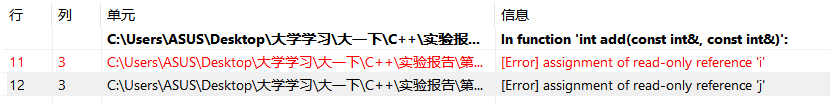
int add(const int &i, const int & j) {

i=i+30;

j=j-10;

return i+j;

}



问题：函数参数为常参数引用 但是函数体中却对常量进行了修改，故编译错误。

2、完善代码中

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 {

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; }**

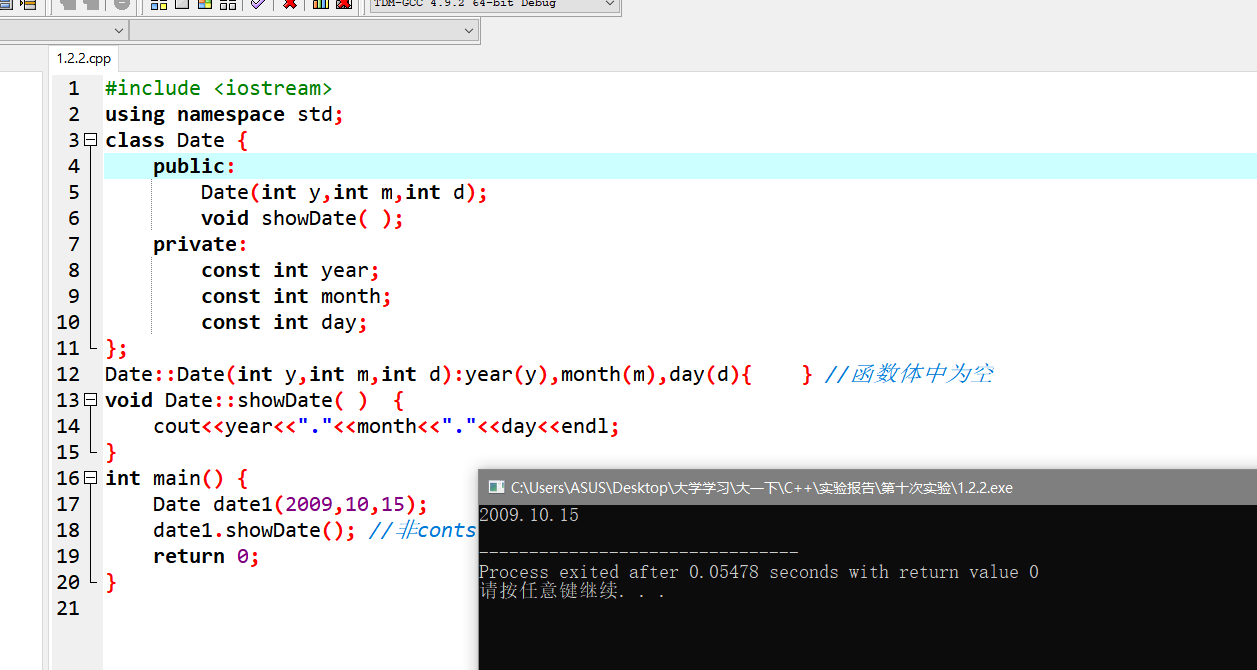
int main() {

Date date1(2009,10,15);

date1.showDate(); //非contst成员函数可以访问const的数据成员

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？

当变量或对象的值是不需要被更新的时候，为了避免后面在代码中写出更改其值的操作，便将其描述为const。

1. 若 **cout<<year<<”.”<<month<<”.”<<day<<endl;** 语句修改为

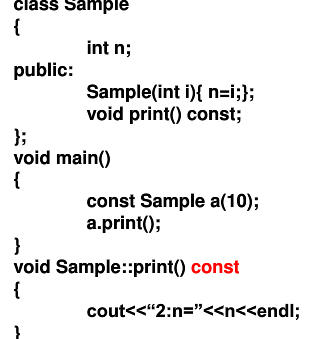
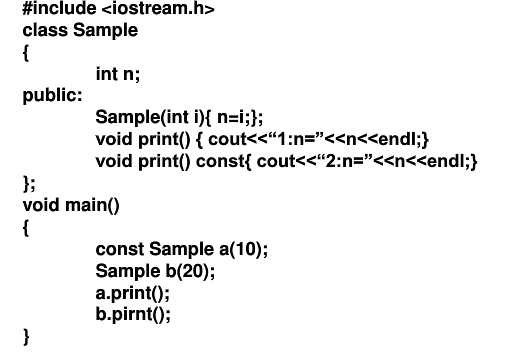
**year=year+1; month=month+1; day=day+1;** 会有什么结果？

将编译错误。对象数据成员为常成员，不能进行修改操作。

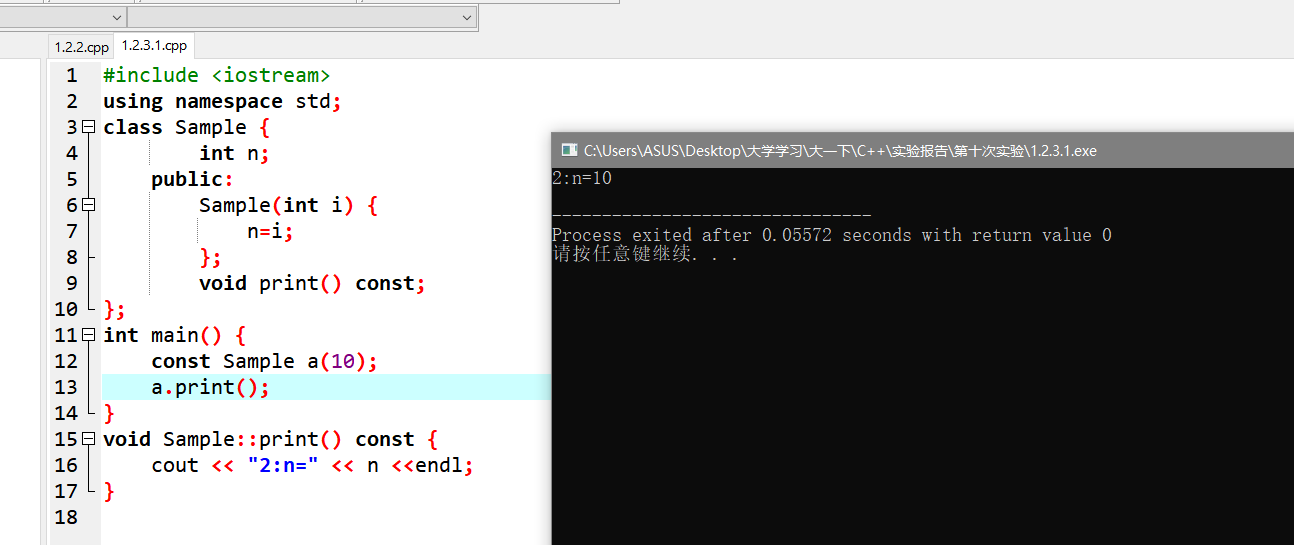
3、测试下列代码（即，针对代码，修改进行，验证性测试。给出界面截图）

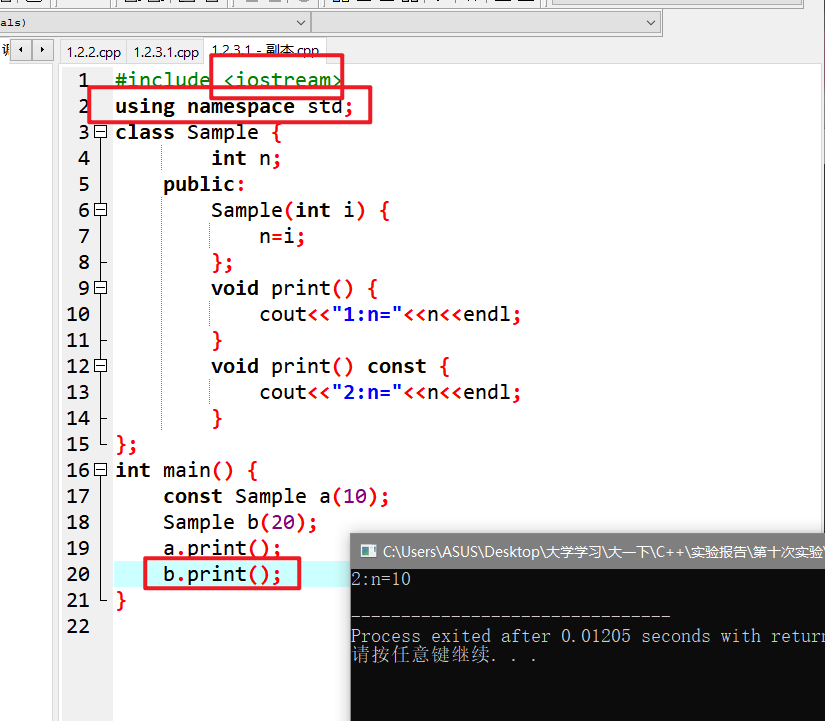
（1）关注：常成员函数的特性

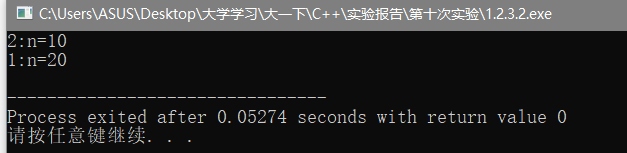
**特性分析：声明和定义都需要写const，且可以作为重载的识别信息；**

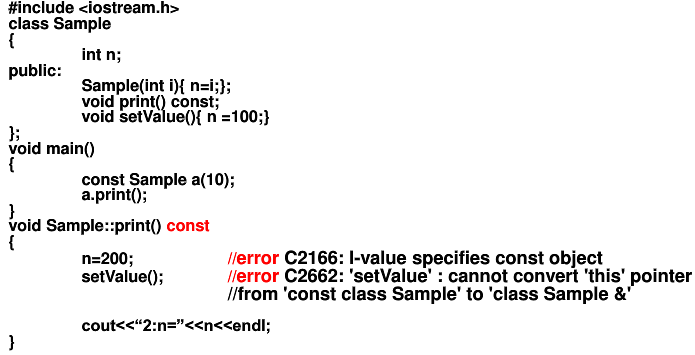
实验过程：



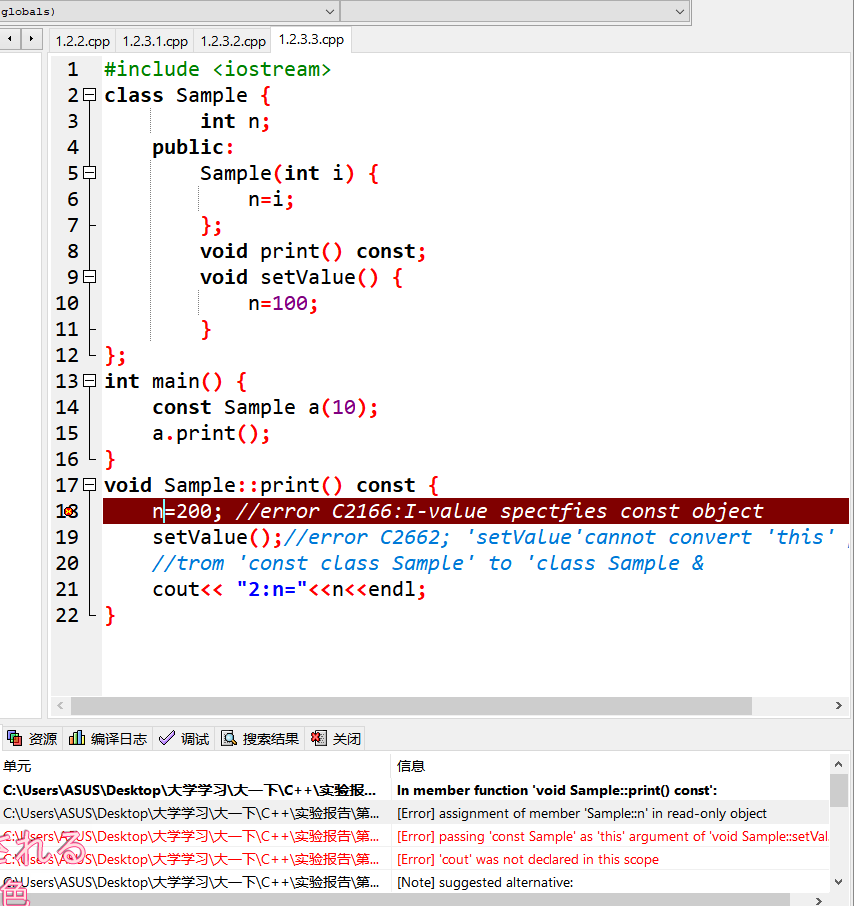
**（红色为修改处）**

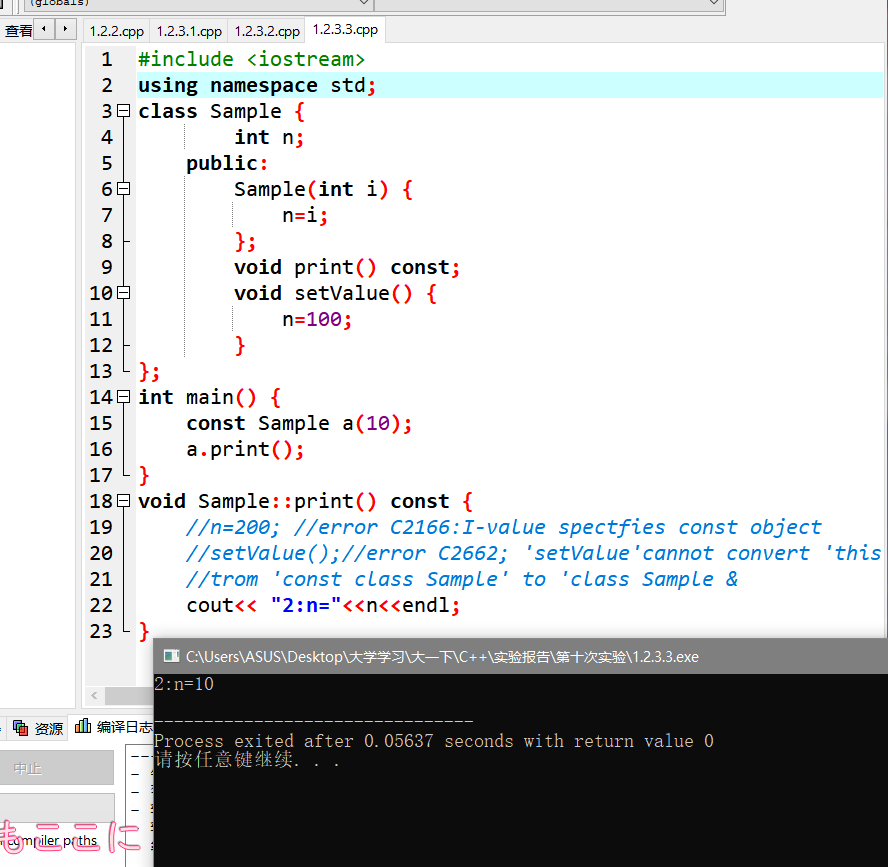
（实际输出结果）

**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；**

****

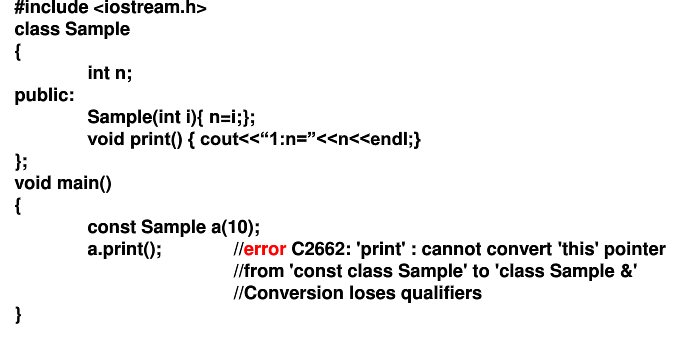
**测试：**

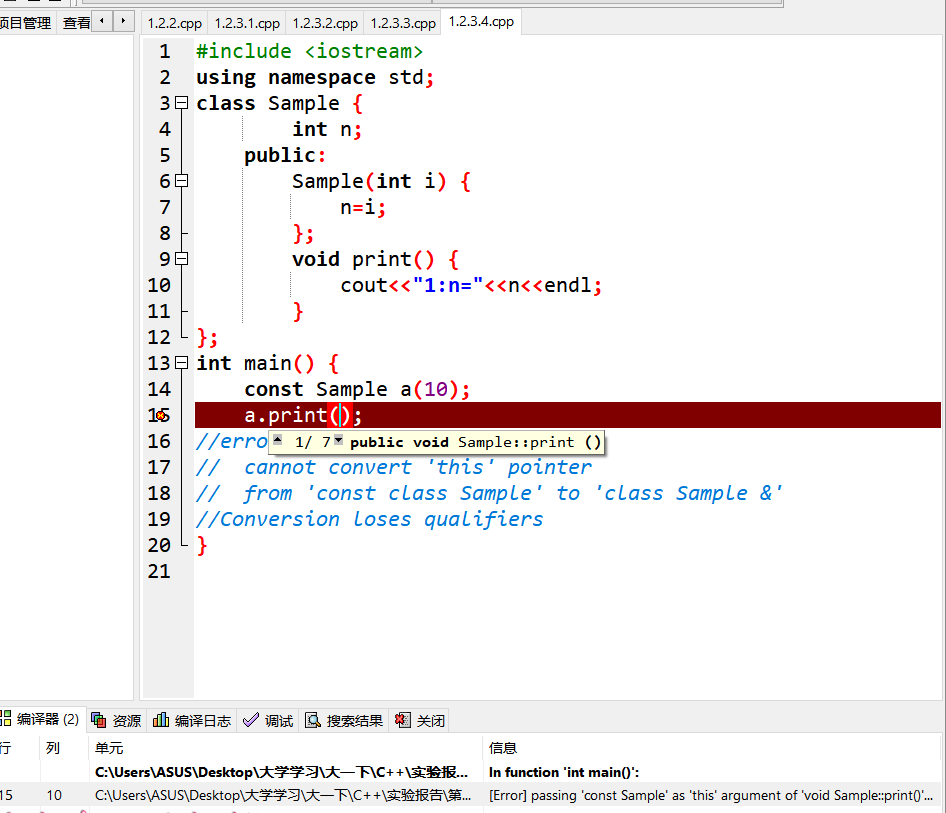


**（注释掉错误代码后即可编译）**

（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**

****



**二、关于友元专题**

1. 选择与简答题
2. 什么是友元机制？友元有哪些分类？

**友元提供了一种 普通函数或者类成员函数 访问另一个类中的私有或保护成员 的机制。也就是说有两种形式的友元：**

**（1）友元函数：普通函数对一个访问某个类中的私有或保护成员。**

**（2）友元类：类A中的成员函数访问类B中的私有或保护成员。**

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 错误，友元函数应当通过对象名和成员运算符访问对象中的成员或是通过对象的引用变量访问对象中的成员等方法，而不是this指针。

（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(Time &t);**

void prin();

private:

int hour;

int minute;

int sec;

};

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

hour=h;

minute=m;

sec=s;

}

void Time:: prin(){

cout<<hour<<minute<<sec;

}

void display(**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

**Display是作为友元函数的形式访问对象里的私有成员，prin是作为类里的函数访问成员。**

（2）请分析：display()函数的参数和函数体的书写特点

**由于友元函数没有this指针，参数需要写上访问的对象的指针或是引用，在函数体中应当通过对象名和成员运算符访问对象中的成员或是通过对象的引用变量访问对象中的成员等方法**

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(){}**

Complex(double r, double i) { real=r;

im=i;

}

Complex(Complex &t) { real=t.real;

im=t.im;

cout<<"HAHA ";

}

**Complex Add(Complex &a) { //要求完善代码，实现复数的加**

**Complex tmp(a.real+real,a.im+im);**

**return tmp;**

**}**

**friend Complex Add2(Complex &a,Complex &b);**

**void display() { cout<<real<<":"<<im; }**

};

**Complex Add2(Complex &a,Complex &b) { //要求完善代码，实现复数的加**

**Complex tmp(a.real+b.real,a.im+b.im);**

**return tmp;**

**}**

int main() {

Complex p1(11,12),p2(13,14);

Complex c1,c2;

c1=p1.Add(p2);

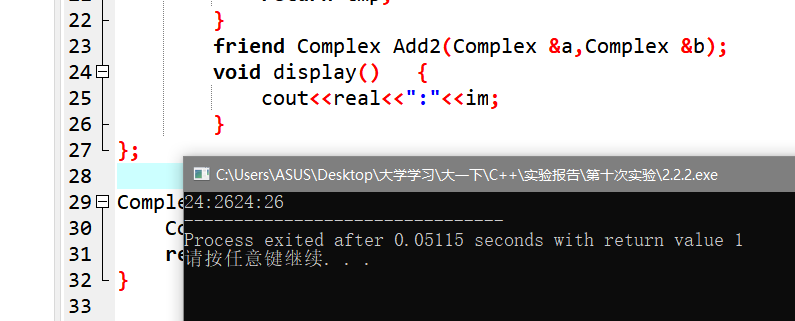
c2=Add2(p1,p2);

c1.display();

c2.display();

return 1;

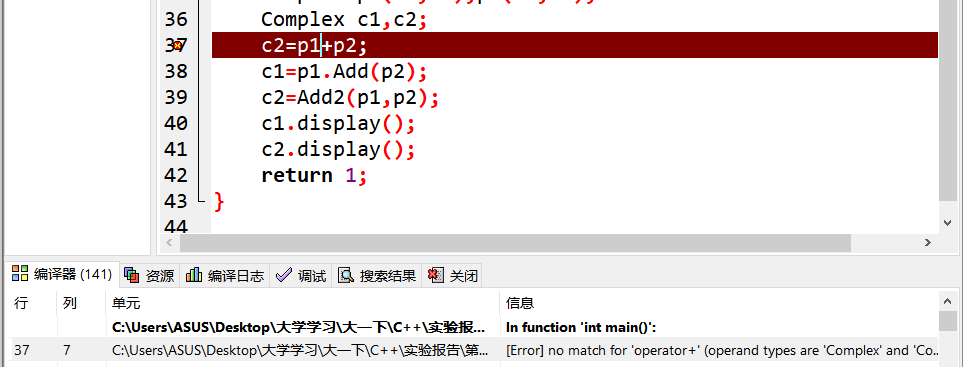
}



1. 根据add和add2位置特性，完善代码；

**如上。**

1. 在main中添加语句：c2= p1+p2; 程序的执行情况反馈

**报错 需要运算符重载**

（3）对main函数中下列语句，重点分析执行过程

**c1=p1.Add(p2);**

**该句调用了成员函数Add();调用this指针指向的p1以及传入的p2相加返回正确的值。**

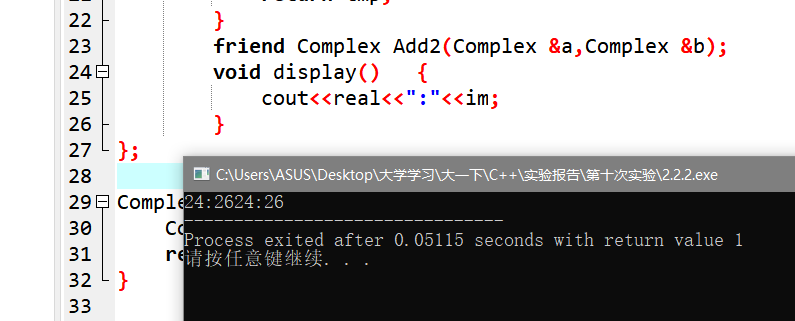
**c2=Add2(p1,p2);**

**该句调用了友元函数Add2();调用参数传入的的p1以及p2相加返回正确的值。**

1. 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？

**可以进行运算符重载。**

1. 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？



**通过本机实验发现并没有调用赋值构造函数（可能原因是c++的优化导致返回值并不会生成复制的对象）**

（三）改错题

1. #include <iostream>
2. class Animal;
3. void setvalue(Animal&,int);
4. void setvalue(Animal&,int,int);
5. class Animal {
   1. public:
   2. friend void Setvalue(Animal &,int);
   3. friend void Setvalue(Animal &,int,int);
   4. protected:
   5. int itsWeight;
   6. int itsAge;
6. };
7. void Setvalue(Animal& ta,int tw) {
   1. ta.itsWeight=tw;
8. }
9. void Setvalue(Animal& ta,int tw,int tn) {
   1. ta.itsWeight=tw;
   2. ta.itsAge=tn;
10. }
11. int main() {
    1. Animal peppy;
    2. Setvalue(peppy,5);
    3. Setvalue(peppy,7,9);
    4. return 0;
12. }

要求：

1. 对错误的代码做修改并标注

如上。

1. 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】

#include <iostream>

class Animal;

void setValue(Animal&,int);

void setValue(Animal&,int,int);

class Animal {

public:

//friend void Setvalue(Animal &,int);

//friend void Setvalue(Animal &,int,int);

**int\* getweight()const{**

**return const\_cast<int\*>(&itsWeight);**

**}**

**int\* getAge()const{**

**return const\_cast<int\*>(&itsAge);**

**}**

protected:

int itsWeight;

int itsAge;

};

void Setvalue(Animal& ta,int tw) {

\*ta.getweight()=tw;

}

void Setvalue(Animal& ta,int tw,int tn) {

\*ta.getweight()=tw;

\*ta.getAge()=tn;

}

int main() {

Animal peppy;

Setvalue(peppy,5);

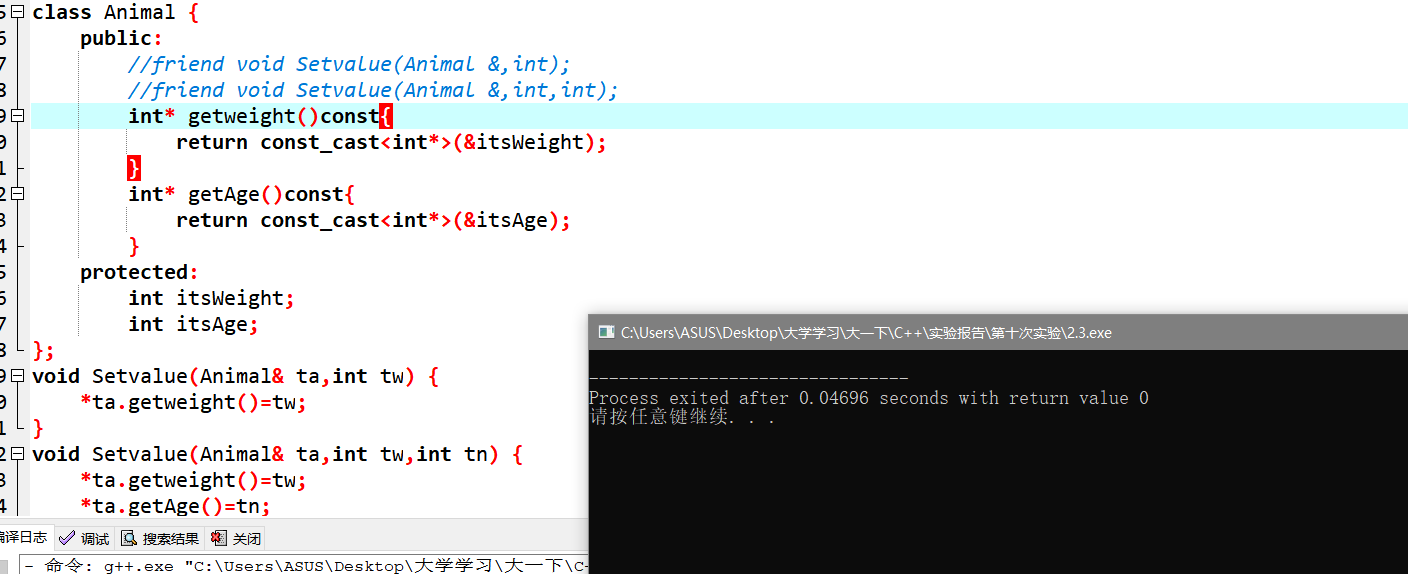
Setvalue(peppy,7,9);

return 0;

}

分析：利用const函数返回指针的形式调用指针里的地址来修改数据。（但破坏了封装性）

实验：成功。



（四）代码题：计算二维坐标系中，两点之间的距离。

类的声明：

class Point {

double x,y;

public:

Point(double xx,double yy):**x(xx),y(yy)** { }

**friend double Distance(Point a,Point b);** //计算两点距离的普通函数

double NewDistance(Point b) {

//计算两点距离的成员函数

**return sprt((b.x-x)\*(b.x-x)+(b.y-y)\*(b.y-y));**

}

};

double Distance(Point a,Point b) {

**return sprt((b.x-a.x)\*(b.x-a.x)+(b.y-a.y)\*(b.y-a.y));**

} (1) 完善类的声明 如上。

(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

double Distance(Point a,Point b) {

**return sprt((b.x-a.x)\*(b.x-a.x)+(b.y-a.y)\*(b.y-a.y));**

}

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

NewDistance**调用了成员函数**NewDistance **();调用this指针指向的a以及传入的b计算返回正确的值。**

**c2=Add2(p1,p2);**

Distance**调用了友元函数**Distance **();调用参数传入的的a以及b相加返回正确的值。**

(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员前提下，仍实现两点之间距离的计算，该如何实现？（思路：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）

示例：

**int getx(){**

**return x;**

**}**

**在外部函数得到x的值，从而进行计算。**

三、**静态成员专题**

（一）简答与选择题

1、静态成员的分类？

**有静态成员变量与静态成员函数。**

1. 静态数据成员有什么特性？如何声明、定义和初始化、使用？

　　1、静态数据成员在定义或说明时前面加关键字static。

　　2、静态成员初始化与一般数据成员初始化不同。静态数据成员初始化的格式如下：

　　　　 <数据类型> <类名> :: <静态数据成员名> = <值>

　　这表明：

　　(1) 初始化在类体外进行，而前面不加static，以免与一般静态变量或对象相混淆。

　　(2) 初始化时不加该成员的访问权限控制符private，public等。

　　(3) 初始化时使用作用域运算符来标明它所属类，因此，静态数据成员是类的成员，而不是对象的成员。

　　3、静态数据成员是静态存储的，它是静态生存期，必须对它进行初始化。

　　4、引用静态数据成员时，采用如下格式：

　　　 <类名> :: <静态成员名>

　　如果静态数据成员的访问权限允许的话(即public的成员)，可在程序中，按上述格式来引用静态数据成员。

3、静态成员函数有什么特性？如何声明、定义和调用？

静态成员函数的特点：

1、被类的所有的对象共有，不属于某一个对象。通过类名::就可以直接调用。

2、跟普通的成员函数比，没有隐藏的this指针作为参数。这一点可用于封装线程类。

3、静态成员函数只可以访问静态成员变量。

4、静态局部变量保存在全局数据区，而不是保存在栈中，每次的值保持到下一次调用，直到下次赋新值。

声明：static 函数返回值类型 函数名（参数1，参数2，…）;

定义：函数返回值类型 类名::函数名（参数1，参数2，…）

{

…

}

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 在体外

B 静态数据成员不可以被类的对象调用 可以

C 静态数据成员年不受private控制符作用 受到

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

|  |  |
| --- | --- |
| 1. 画出y空间的结构示意图，分析结果     过程如上。  #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=”<<endl; }  };  **int Sample::y=10;**  void main() {  Sample s1(20);  Sample s2(30);  s1.print(); s2.print();  }     1. 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。     将编译失败。静态变量是私有的，外部不能访问。   1. 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。     可以运行，此时为public公开变量，可以访问。 | 1、分析static成员函数，找出错误分析原因；    静态成员函数只可以访问静态成员变量  故不能调用x  2、用文字总结static成员函数特性  静态成员函数的特点：  1、被类的所有的对象共有，不属于某一个对象。通过类名::就可以直接调用。  2、跟普通的成员函数比，没有隐藏的this指针作为参数。这一点可用于封装线程类。  3、静态成员函数只可以访问静态成员变量。  4、静态局部变量保存在全局数据区，而不是保存在栈中，每次的值保持到下一次调用，直到下次赋新值。  #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);  } |

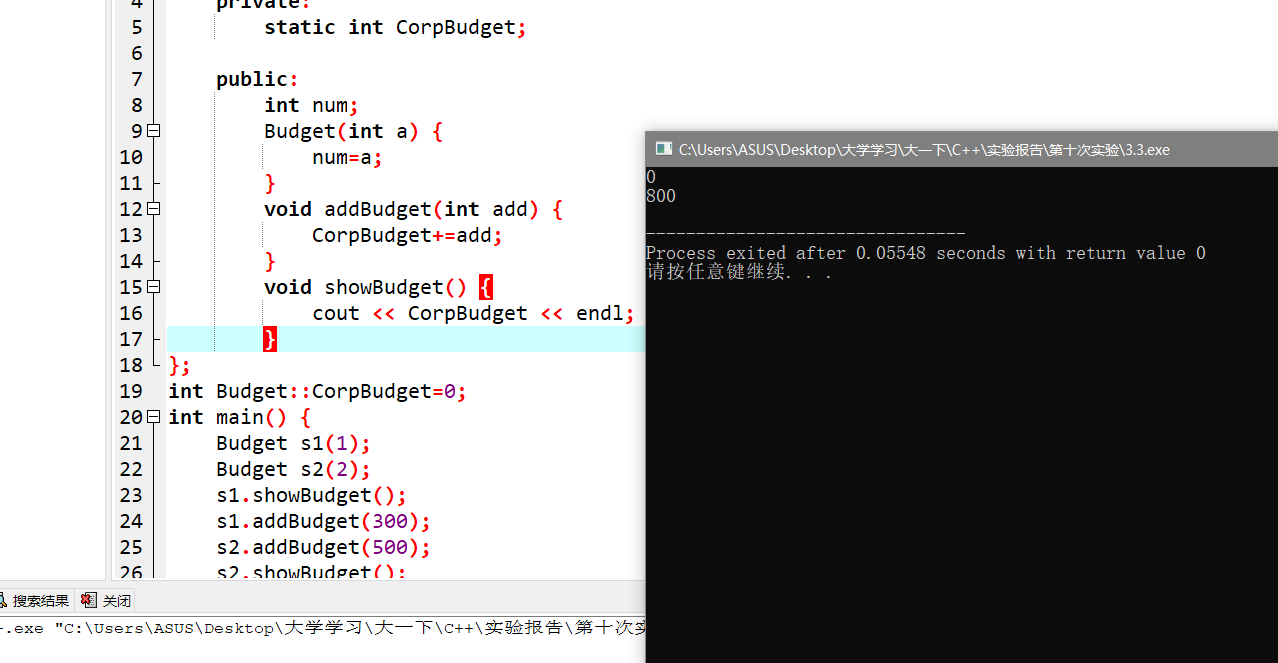
3、代码题

某公司由若干个子公司组成，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。

见文件3.3.cpp



成功实现增加800的预算