第 3 章 怎样使用类和对象

1．构造函数和析构函数的作用是什么？什么时候需要自己定义构造函数和析构函数？

【解】 略。

2．分析下面的程序，写出其运行时的输出结果。

```
#include <iostream>
using namespace std;
class Date
 {public:
    Date(int,int,int);
    Date(int,int);
    Date(int);
    Date( );
    void display( );
  private:
    int month;
    int day;
    int year;
 };

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

Date::Date(int m,int d): month(m),day(d)
 {year=2005;}

Date::Date(int m):month(m)
 {day=1;
  year=2005;
 }

Date::Date( )
```

```
 {month=1;
  day=1;
  year=2005;
 }

void Date::display( )
 {cout<<month<<"/"<<day<<"/"<<year<<endl;}

int main( )
{
   Date d1(10,13,2005);
   Date d2(12,30);
   Date d3(10);
   Date d4;
   d1.display( );
   d2.display( );
   d3.display( );
   d4.display( );
   return 0;
 }
```

【解】 程序运行结果为

```
10/13/2005
12/30/2005
10/1/2005
1/1/2005
```

3．如果将第 2 题中程序的第 4 行改为用默认参数，即

```
Date(int=1,int=1,int=2005);
```

分析程序有无问题。上机编译，分析出错信息，修改程序使之能通过编译。要求保留上面一行给出的构造函数，同时能输出与第 2 题的程序相同的输出结果。

【解】 编译时出错，因为构造函数使用默认参数后就不能再使用重载的构造函数，否则就会出现歧义性，例如在处理

```
Date d2(12,30);
```

时，系统无法辨别应当调用默认参数的构造函数

```
Date(int=1,int=1,int=2005);
```

还是调用重载的构造函数

```
Date(int,int);
```

系统不允许出现这样的矛盾现象，会给出出错信息，要求修改程序。可修改程序如下：

```
#include <iostream>
using namespace std;
class Date
 {public:
    Date(int=1,int=1,int=2005);
    void display( );
  private:
    int month;
    int day;
    int year;
 };

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

void Date::display( )
 {cout<<month<<"/"<<day<<"/"<<year<<endl;}

int main( )
{
 Date d1(10,13,2005);
 Date d2(12,30);
 Date d3(10);
 Date d4;
 d1.display( );
 d2.display( );
 d3.display( );
 d4.display( );
 return 0;
}
```

删掉重载的构造函数，这时再编译，无错误，运行结果同第 2 题。

4．建立一个对象数组，内放 5 个学生的数据（学号、成绩），用指针指向数组首元素，输出第 1,3,5 个学生的数据。

【解】 程序如下：

```
#include <iostream>
using namespace std;
class Student
 {public:
    Student(int n,float s):num(n),score(s){ }
    void display( );
  private:
```

```
    int num;
    float score;
  };

void Student::display( )
  {cout<<num<<″ ″<<score<<endl;}

int main( )
{Student stud[5]={Student(101,78.5),Student(102,85.5),Student(103,98.5), Student(104,100.0),
                  Student(105,95.5)};
  Student *p=stud;
  for(int i=0;i<=2;p=p+2,i++)
    p->display( );
  return 0;
  }
```

运行时的输出如下：

```
101 78.5
103 98,5
105 95.5
```

5．建立一个对象数组，内放 5 个学生的数据(学号、成绩)，设立一个函数 max，用指向对象的指针作函数参数，在 max 函数中找出 5 个学生中成绩最高者，并输出其学号。

【解】 程序如下：

```
#include <iostream>
using namespace std;
class Student
  {public:
    Student(int n,float s):num(n),score(s){ }
    int num;
    float score;
  };

int main( )
{Student stud[5]={ Student(101,78.5),Student(102,85.5),Student(103,98.5),
      Student(104,100.0),Student(105,95.5)};
  void max(Student* );
  Student *p=&stud[0];
  max(p);
  return 0;
  }
void max(Student *arr)
```

```
{float max_score=arr[0].score;
 int k=0;
 for(int i=1;i<5;i++)
    if(arr[i].score>max_score) {max_score=arr[i].score;k=i;}
 cout<<arr[k].num<<" "<<max_score<<endl;
}
```

6．阅读下面程序，分析其执行过程，写出输出结果。

```
#include <iostream>
using namespace std;
class Student
 {public:
    Student(int n,float s):num(n),score(s){ }
    void change(int n,float s) {num=n;score=s;}
    void display( ) {cout<<num<<" "<<score<<endl;}
  private:
    int num;
    float score;
 };

int main( )
{Student stud(101,78.5);
 stud.display( );
 stud.change(101,80.5);
 stud.display( );
 return 0;
}
```

【解】 函数 stud.display 的作用是输出对象 stud 中数据成员 num 和 score 的值，函数 stud.change 的作用是改变对象 stud 中数据成员 num 和 score 的值，在调用此函数时给出实参 101 和 80.5，取代了数据成员 num 和 score 原有的值。

程序运行结果如下：

```
101 78.5                    （num 和 score 的原值）
101 80.5                    （num 和 score 的新值）
```

7．将第 6 题的程序分别作以下修改，分析所修改部分的含义以及编译和运行的情况。

（1）将 main 函数中的第 2 行改为

```
const Student stud(101,78.5);
```

（2）在（1）的基础上修改程序，使之能正常运行，用 change 函数修改数据成员 num 和 score 的值。

（3）将 main 函数改为

```
int main( )
{Student stud(101,78.5);
 Student *p=&stud;
 p->display( );
 p->change(101,80.5);
 p->display( );
 return 0;
}
```

其他部分仍同第 6 题的程序。

（4）在（3）的基础上将 main 函数第 3 行改为

```
const   Student *p=&stud;
```

（5）再把 main 函数第 3 行改为

```
Student *const p=&stud;
```

【解】

（1）有两个错误：

① stud 被声明为常对象后，不能调用对象中的一般成员函数（除非把该成员函数也声明为 const 型），因此在 main 函数中调用 stud.display( )和 stud.change( )是非法的。

② 若将对象 stud 声明为常对象，其值是不可改变的，而在主程序中，企图用 stud.change 函数去改变 stud 中数据成员的值，是非法的。

因此程序在编译时出错。如果将程序第 7 行改为

```
void display( ) const {cout<<num<<" "<<score<<endl;}
```

把 display 函数改为 const 型，可以正常调用 display 函数。如果把第 6 行也改为

```
void change(int n,float s) const {num=n;score=s;}
```

程序编译时仍然出错，这是由于 change 函数企图改变 stud 中数据成员的值。如果删去 main 函数中调用 change 函数的一行（可把它改为注释行），则程序能通过编译，可以正常运行。读者可以自己上机调试一下。

（2）要求用 change 函数修改数据成员 num 和 score 的值，则将数据成员 num 和 score 声明为可变的（mutable）数据成员即可。程序如下：

```
#include <iostream>
using namespace std;
class Student
 {public:
    Student(int n,float s):num(n),score(s){ }
    void change(int n,float s) const    {num=n;score=s;}                    //常成员函数
    void display( )const {cout<<num<<" "<<score<<endl;}                     //常成员函数
```

```
  private:
    mutable int num;                          //用 mutable 声明可变的数据成员
    mutable float score;                      //用 mutable 声明可变的数据成员
};

int main( )
{const Student stud(101,78.5);                //常对象
 stud.display( );                             //调用常成员函数
 stud.change(101,80.5);                       //调用常成员函数，修改数据成员
 stud.display( );
 return 0;
}
```

程序运行结果如下：

```
101 78.5                                      （修改前的数据）
101 80.5                                      （修改后的数据）
```

（3）根据题目要求，程序改为

```
#include <iostream>
using namespace std;
class Student
 {public:
    Student(int n,float s):num(n),score(s){ }
    void change(int n,float s) {num=n;score=s;}
    void display( ) {cout<<num<<" "<<score<<endl;}
  private:
    int num;
    float score;
 };

int main( )
{Student stud(101,78.5);
 Student *p=&stud;
 p->display( );
 p->change(101,80.5);
 p->display( );
 return 0;
}
```

在主函数中定义了指针对象 p，它指向 stud，函数 p->display( )相当于 stud.display( )。程序合法，运行结果与第 6 题的程序的运行结果相同。

（4）在（3）的基础上将 main 函数第 3 行改为

```
const   Student *p=&stud;
```

程序如下：

```
#include <iostream>
using namespace std;
class Student
 {public:
    Student(int n,float s):num(n),score(s){ }
    void change(int n,float s) {num=n;score=s;}
    void display( ) {cout<<num<<″ ″<<score<<endl;}
  private:
    int num;
    float score;
 };

int main( )
{Student stud(101,78.5);
 const Student *p=&stud;
 p->display( );
 p->change(101,80.5);
 p->display( );
 return 0;
}
```

在主函数中定义了指向 const 对象的指针变量 p，则其指向的对象的值是不能通过指针变量 p 改变的。为了安全，C++也不允许通过指针变量 p 调用对象 stud 中的非 const 成员函数，在 main 函数中调用 p->display( )和 p->change( )是非法的。为了能正确调用 stud 中的 display 函数，应将程序第 7 行改为

```
void display( ) const {cout<<num<<″ ″<<score<<endl;}
```

即将 display 函数声明为 const 型。这样保证 display 函数只能引用而不能修改类中的数据成员。

此外，p->change( )企图通过指针变量 p 修改类中的数据成员的值，这也是和指向 const 对象的指针变量的性质不相容的，编译时出错。

在上面的基础上，将程序改为

```
#include <iostream>
using namespace std;
class Student
 {public:
    Student(int n,float s):num(n),score(s){ }
    void change(int n,float s) {num=n;score=s;}
    void display( ) const{cout<<num<<″ ″<<score<<endl;}     //此行加了 const
  private:
    int num;
```

```
    float score;
 };

int main( )
{Student stud(101,78.5);
 const Student *p=&stud;
 p->display( );
 stud.change(101,80.5);                    //注意此行修改了
 p->display( );
 return 0;
}
```

在 main 函数中，不是通过指针变量 p 修改数据成员，而直接通过对象名 stud 调用 change 函数，则是允许的，编译能通过。因为并未定义 stud 为常对象，只是定义了 p 是指向 const 对象的指针变量，不能通过指针变量 p 修改类中的数据成员的值，而不通过指针变量 p 修改类中的数据成员的值是可以的。

同样，如果不是通过指针变量 p 调用 display 函数（即 p->display( );），而是通过对象名 stud 调用 display 函数，则不必将 display 函数声明为 const 型。

（5）再把 main 函数第 3 行改为

```
Student *const p=&stud;
```

定义了一个指向对象的常指针，要求指针变量 p 的指向不能改变，只能始终指向对象 stud。今在程序中未改变 p 的指向，因此程序合法，而且不需要在定义 display 和 change 函数时将它们声明为 const。程序能通过编译，并正常运行。运行的结果与第 6 题的程序的运行结果相同。

8．修改第 6 题的程序，增加一个 fun 函数，改写 main 函数。在 main 函数中调用 fun 函数，在 fun 函数中调用 change 和 display 函数。在 fun 函数中使用对象的引用（Student &）作为形参。

【解】 可以编写出以下程序：

```
#include <iostream>
using namespace std;
class Student
 {public:
    Student(int n,float s):num(n),score(s){ }
    void change(int n,float s) {num=n;score=s;}
    void display( ) {cout<<num<<" "<<score<<endl;}
  private:
    int num;
    float score;
 };

int main( )
```

```
{Student stud(101,78.5);
 void fun(Student &);                    //声明 fun 函数
 fun(stud);                              //调用 fun 函数，实参为对象 stud
 return 0;
}

void fun(Student &stu)                   //定义 fun 函数，形参为 Student 类对象的引用
{stu.display( );                         //在 fun 函数中调用 change 和 display 函数
 stu.change(101,80.5);
 stu.display( );

}
```

运行结果如下：

```
101 78.5
101 80.5
```

9．商店销售某一商品，商店每天公布统一的折扣（discount）。同时允许销售人员在销售时灵活掌握售价（price），在此基础上，对一次购 10 件以上者，还可以享受 9.8 折优惠。现已知当天 3 个销货员的销售情况为：

销货员号（num）	销货件数（quantity）	销货单价（price）
101	5	23.5
102	12	24.56
103	100	21.5

请编程序，计算出当日此商品的总销售款 sum 以及每件商品的平均售价。要求用静态数据成员和静态成员函数。

（提示：将折扣 discount，总销售款 sum 和商品销售总件数 n 声明为静态数据成员，再定义静态成员函数 average（求平均售价）和 display（输出结果）。

【解】 可以编写出以下程序：

```
#include <iostream>
using namespace std;
class Product
 {public:
    Product(int m,int q,float p):num(m),quantity(q),price(p){ };
    void total( );
    static float average( );
    static void display( );
  private:
    int num;                             //销货员号
    int quantity;                        //销货件数
    float price;                         //销货单价
    static float discount;               //商店统一折扣
```

```
    static float sum;                                   //总销售款
    static int n;                                       //商品销售总件数
};

void Product::total( )                                  //求销售款和销售件数
 {float rate=1.0;
  if(quantity>10) rate=0.98*rate;
  sum=sum+quantity*price*rate*(1–discount);             //累计销售款
  n=n+quantity;                                         //累计销售件数
 }

void Product::display( )                                //输出销售总件数和平均价
 {cout<<sum<<endl;
  cout<<average( )<<endl;
 }

float Product::average( )                               //求平均价
 {return(sum/n);}

float Product::discount=0.05;                           //对静态数据成员初始化
float Product::sum=0;                                   //对静态数据成员初始化
int Product::n=0;                                       //对静态数据成员初始化

int main( )
 { Product Prod[3]={Product(101,5,23.5),Product(102,12,24.56),Product(103,100,21.5)};
                                                        //定义 Product 类对象数组，并给出数据
   for(int i=0;i<3;i++)                                 //统计 3 个销货员的销货情况
      Prod[i].total( );
   Product::display( );                                 //输出结果
   return 0;
 }
```

运行结果如下：

```
2387.66                                 （总销售款）
20.4073                                 （平均售价）
```

读者可以在此基础上对输出结果做一些加工和修饰，如加上必要的文字说明，对输出的数值取两位小数等。

10．将《C++面向对象程序设计（第 2 版）》第 3 章例 3.13 程序中的 display 函数不放在 Time 类中，而作为类外的普通函数，然后分别在 Time 和 Date 类中将 display 声明为友元函数。在主函数中调用 display 函数，display 函数分别引用 Time 和 Date 两个类的对象的私有数据，输出年、月、日和时、分、秒。请读者自己完成并上机调试。

【解】 可以编写出以下程序：

```
#include <iostream>
using namespace std;
class Date;                                          //对 Date 的声明，它是对 Date 的预引用
class Time
{public:
    Time(int,int,int);
    friend void display(const Date &,const Time &);  //将普通函数 display 声明为朋友
  private:
     int hour;
     int minute;
     int sec;
 };

 Time::Time(int h,int m,int s)
   {hour=h;
    minute=m;
    sec=s;
   }

class Date
 {public:
    Date(int,int,int);
    friend void display(const Date &,const Time &);  //将普通函数 display 声明为朋友
  private:
     int month;
     int day;
     int year;
 };

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

void display(const Date &d,const Time &t)            //是 Time 和 Date 两个类的朋友
 {
  cout<<d.month<<"/"<<d.day<<"/"<<d.year<<endl;      //引用 Date 类对象 t1 中的数据成员
  cout<<t.hour<<":"<<t.minute<<":"<<t.sec<<endl;     //引用 Time 类对象 t1 中的数据成员
 }

int main( )
```

```
{
 Time t1(10,13,56);                    //定义 Time 类对象 t1
 Date d1(12,25,2004);                  //定义 Date 类对象 d1
 display(d1,t1);                       //调用 display 函数，用对象名作实参
 return 0;
}
```

运行结果如下：

```
12/25/2004
10:13:56
```

11．将《C++面向对象程序设计（第 2 版）》第 3 章例 3.13 中的 Time 类声明为 Date 类的友元类，通过 Time 类中的 display 函数引用 Date 类对象的私有数据，输出年、月、日和时、分、秒。

【解】 可以编写出以下程序：

```
#include <iostream>
using namespace std;
class Time;
class Date
 {public:
    Date(int,int,int);
    friend Time;                       //将 Time 类声明为朋友类
  private:
     int month;
     int day;
     int year;
 };

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

class Time
 {public:
    Time(int,int,int);
    void display(const Date &);
  private:
     int hour;
     int minute;
     int sec;
 };

Time::Time(int h,int m,int s):hour(h),minute(m),sec(s){ }

void Time::display(const Date &d)
 {
```

```
  cout<<d.month<<"/"<<d.day<<"/"<<d.year<<endl;          //引用 Date 类对象 d1 的数据成员
  cout<<hour<<":"<<minute<<":"<<sec<<endl;              //引用 Time 类对象 d1 的数据成员
 }

int main( )
{
 Time t1(10,13,56);
 Date d1(12,25,2004);
 t1.display(d1);
 return 0;
}
```

运行结果如下：

```
12/25/2004
10:13:56
```

由于 Time 类是 Date 类的友元类，因此 Time 类中的成员函数都是 Date 类的友元函数，它既可以引用 Time 类对象的数据成员，又可以引用 Date 类对象的数据成员。在引用本类(Time 类)的数据成员时，不必在数据成员名前面加对象名，而在引用 Date 类的数据成员时必须在数据成员名前面加上对象名(如 d.month。d 是形参名，实参是对象 d1，因此 d.month 相当于 d1.month)。

12．将《C++面向对象程序设计（第 2 版）》第 3 章例 3.14 改写为在类模板外定义各成员函数。

【解】 改写后的程序如下：

```
#include <iostream>
using namespace std;
template<class numtype>
class Compare
 {public:
    Compare(numtype a,numtype b);
    numtype max( );
    numtype min( );
  private:
     numtype x,y;
 };

//在类模板外定义各成员函数
template <class numtype>
Compare<numtype>::Compare(numtype a,numtype b)
   {x=a;y=b;}

template <class numtype>
numtype Compare<numtype>::max( )
```

```
  {return (x>y)?x:y;}
template <class numtype>
numtype Compare<numtype>::min( )
    {return (x<y)?x:y;}

//主函数
int main( )
{Compare<int> cmp1(3,7);
  cout<<cmp1.max( )<<″ is the Maximum of two integer numbers.″<<endl;
  cout<<cmp1.min( )<<″ is the Minimum of two integer numbers.″<<endl<<endl;
  Compare<float> cmp2(45.78,93.6);
  cout<<cmp2.max( )<<″ is the Maximum of two float numbers.″<<endl;
  cout<<cmp2.min( )<<″ is the Minimum of two float numbers.″<<endl<<endl;
  Compare<char> cmp3(′a′,′A′ );
  cout<<cmp3.max( )<<″ is the Maximum of two characters.″<<endl;
  cout<<cmp3.min( )<<″ is the Minimum of two characters.″<<endl;
  return 0;
}
```

运行结果为

```
7 is the Maximum of two integers.
3 is the Minimum of two integers.

93.6 is the Maximum of two float numbers.
45.78 is the Minimum of two float numbers.

a is the Maximum of two characters.
A is the Minimum of two characters .
```

第 4 章 对运算符进行重载

1．定义一个复数类 Complex，重载运算符“+”，使之能用于复数的加法运算。将运算符函数重载为非成员、非友元的普通函数。编程序，求两个复数之和。

【解】 根据题意，可编程序如下：

```
#include <iostream>
using namespace std;
class Complex
 {public:
     Complex( ){real=0;imag=0;}
     Complex(double r,double i){real=r;imag=i;}
     double get_real( );                          //声明 get_real 函数
     double get_imag( );                          //声明 get_imag 函数
     void display( );
  private:
     double real;
     double imag;
 };

double Complex::get_real( )                       //取数据成员 real（实部）的值
{return real;}

double Complex::get_imag( )                       //取数据成员 imag（虚部）的值
{return imag;}

void Complex::display( )
{cout<<"("<<real<<","<<imag<<"i)"<<endl;}

Complex operator+(Complex &c1,Complex &c2)
{return Complex(c1.get_real( )+c2.get_real( ),c1.get_imag( )+c2.get_imag( ));}

int main( )
{Complex c1(3,4),c2(5,–10),c3;
 c3=c1+c2;
```

```
 cout<<"c3=";
 c3.display( );
 return 0;
}
```

运行结果为

```
c3=(8,– 6i)
```

说明：

（1）运算符重载函数既不是类 Complex 的成员函数，也不是类 Complex 的友元函数，而是一个普通函数。

（2）由于运算符重载函数 operator+是非成员和非友元的普通函数，因此它不能直接引用 Complex 类中的私有成员，在此函数中以下的写法是错误的：

```
return Complex(c1.real+c2.real,c1.imag+c2.imag);
```

只能通过 Complex 类中的公用函数 get_real 和 get_imag 去引用类的私有成员 real 和 imag。

（3）公用函数 get_real 和 get_imag 的类型是 double，get_real( )的返回值是 real，get_imag( )的返回值是 imag。因此，c1.get_real( )+c2.get_real( )就相当于 c1.real + c2.real，而 c1.get_imag( )+c2.get_imag( )就相当于 c1.imag+c2.imag。运算符重载函数的返回值是两个复数 c1 和 c2 之和。

（4）虽然程序能正常运行，且结果正确，但这个程序并不是理想的。请看引用 Complex 类对象私有成员的步骤：在主函数中用 c1+c2 调用运算符重载函数 operator+ → 在运算符重载函数中调用 Complex 类的公用函数 get_real 和 get_imag → get_real 和 get_imag 引用本类中的私有成员。兜了一个圈子，很不直观，很不方便。显然这种方法不如将运算符函数重载为成员函数和友元函数方便。

（5）本习题的目的是：①学习编写基于对象的程序，提高编程能力，学会在不同的情况下找到解决问题的方法；②更重要的是比较运算符函数重载的不同方法，得到一个结论：一般不把运算符函数重载为非成员和非友元的普通函数。

2．定义一个复数类 Complex，重载运算符“+”，“–”，“*”，“/”，使之能用于复数的加、减、乘、除。运算符重载函数作为 Complex 类的成员函数。编程序，分别求两个复数之和、差、积和商。

【解】 从数学知识可知：

如果有两个复数：$m = a + bi, n = c + di$。复数的加、减、乘、除的公式如下：

（1）复数加法：$m + n = a + bi + c + di = (a + c) + (b + d)i$

（2）复数减法：$m - n = a + bi - c - di = (a - c) + (b - d)i$

（3）复数乘法：$m \times n = (a + bi)(c + di) = ac + bci + adi + bdi^2 = (ac - bd) + (bc + ad)i$

（4）复数除法：

$$\frac{m}{n} = \frac{a+bi}{c+di} = \frac{(a+bi)(c-di)}{(c+di)(c-di)} = \frac{ac+bci-adi-bdi^2}{c^2+d^2} = \frac{(ac+bd)+(bc-ad)i}{c^2+d^2}$$

$$=\frac{ac+bd}{c^2+d^2}+\frac{bc-ad}{c^2+d^2}i$$

根据以上公式，可以写出以下程序：

```
#include <iostream>
using namespace std;
class Complex
  {public:
      Complex( ){real=0;imag=0;}
      Complex(double r,double i){real=r;imag=i;}
      Complex operator + (Complex &c2);
      Complex operator – (Complex &c2);
      Complex operator*(Complex &c2);
      Complex operator/(Complex &c2);
      void display( );
   private:
      double real;
      double imag;
 };

Complex Complex::operator + (Complex &c2)              //重载运算符“+”
{Complex c;
 c.real=real+c2.real;                                  //计算实部
 c.imag=imag+c2.imag;                                  //计算虚部
 return c;}

Complex Complex::operator– (Complex &c2)               //重载运算符“–”
{Complex c;
 c.real=real– c2.real;                                 //计算实部
 c.imag=imag– c2.imag;                                 //计算虚部
 return c;}


Complex Complex::operator*(Complex &c2)                //重载运算符“*”
{Complex c;
 c.real=real*c2.real–imag*c2.imag;                     //计算实部
 c.imag=imag*c2.real+real*c2.imag;                     //计算虚部
 return c;}

Complex Complex::operator/(Complex &c2)                //重载运算符“/”
{Complex c;
 c.real=(real*c2.real+imag*c2.imag)/(c2.real*c2.real+c2.imag*c2.imag);          //计算实部
 c.imag=(imag*c2.real–real*c2.imag)/(c2.real*c2.real+c2.imag*c2.imag);          //计算虚部
 return c;}
```

```
void Complex::display( )
{cout<<"("<<real<<","<<imag<<"i)"<<endl;}        //输出复数

int main( )
{Complex c1(3,4),c2(5,–10),c3;
 c3=c1+c2;
 cout<<"c1+c2=";
 c3.display( );
 c3=c1– c2;
 cout<<"c1– c2=";
 c3.display( );
 c3=c1*c2;
 cout<<"c1*c2=";
 c3.display( );
 c3=c1/c2;
 cout<<"c1/c2=";
 c3.display( );
 return 0;
}
```

运行结果如下：

```
c1 + c2=(8,– 6i)
c1 – c2=(–2,14i)
c1*c2=(55,–10i)
c1/c2=(– 0.2,0.4i)
```

3．定义一个复数类 Complex，重载运算符“+”，使之能用于复数的加法运算。参加运算的两个运算量可以都是类对象，也可以其中有一个是整数，顺序任意。例如：c1+c2，i+c1，c1+i 均合法（设 i 为整数，c1，c2 为复数）。编程序，分别求两个复数之和、整数和复数之和。

【解】 程序如下：

```
#include <iostream>
using namespace std;
class Complex
 {public:
    Complex( ){real=0;imag=0;}
    Complex(double r,double i){real=r;imag=i;}
    Complex operator+(Complex &c2);                    //运算符重载为成员函数
    Complex operator+(int &i);                         //运算符重载为成员函数
    friend Complex operator+(int&,Complex &);          //运算符重载为友元函数
    void display( );
  private:
    double real;
```

```
    double imag;
};

Complex Complex::operator+(Complex &c)                    //定义成员运算符函数
{return Complex(real+c.real,imag+c.imag);}

Complex Complex::operator+(int &i)                        //定义成员运算符函数
{return Complex(real+i,imag);}

void Complex::display( )
{cout<<"("<<real<<","<<imag<<"i)"<<endl;}

Complex operator+(int &i,Complex &c)                      //定义友元运算符函数
{return Complex(i+c.real,c.imag);}

int main( )
{Complex c1(3,4),c2(5,–10),c3;
 int i=5;
 c3=c1+c2;
 cout<<"c1+c2=";
 c3.display( );
 c3=i+c1;
 cout<<"i+c1=";
 c3.display( );
 c3=c1+i;
 cout<<"c1+i=";
 c3.display( );
 return 0;
}
```

运行结果如下：

```
c1+c2=(8,– 6i)
i+c1=(8,4i)
c1+i=(8,4i)
```

在程序中对运算符“+”进行了 3 次重载，分别是

```
Complex operator+(Complex &,Complex &);        //形参为类对象和类对象
Complex operator+(Complex &,int &);            //形参为类对象和整型数
Complex operator+(int &,Complex &);            //形参为整型数和类对象
```

由于前两个重载函数的第一个参数为类对象，所以将它们作为类的成员函数，函数的第一个参数也可以省略。第 3 个重载函数的第一个参数为 int 型，不是类对象，不能作为类的成员函数，只能作为友元函数，函数的两个参数不能省略。