第5章 类与对象

结构化程序设计的重点在于函数设计，而函数就是程序模块的基本功能单元，是对待处理任务的一种抽象。把一切逻辑功能完全独立的或相对独立的程序部分都设计成函数，并让每一个函数只完成单一的功能。这样，一个函数就是一个程序模块，程序的各个部分除了必要的信息交流之外，互不影响。函数之间的相互调用，构成了完整的可运行的系统。

随着软件复杂性的不断提高，把过程化或者结构化概念应用于大型的、复杂的程序中可能导致各种各样的问题，比如：

(1) 难以维护和修改程序。

(2) 许多编程细节难以组织，增加了程序员的负担。

(3) 难以调试程序并跟踪其逻辑。

(4) 导致诸如意外数据修改等逻辑错误的产生。

面向对象程序设计不是以函数过程和数据结构为中心，而是以对象代表求解问题的中心环节，它追求的是现实问题空间与软件系统空间的近似和直接模拟。面向对象程序设计的方式与人类社会认识和理解客观事物的思路是高度一致的。在客观世界和社会生活中，复杂的事物总是由许多部分组成的。人们生产一台计算机时，总是分别设计和制造显示器、键盘、中央处理器、显示卡、主板、内存、硬盘、电源等，最后把它们组装成一台计算机。在组装时，各部分之间有一定的联系和兼容的标准，以便协调工作。这就是面向对象程序设计的基本思路。

在面向对象程序设计中，程序模块由类构成。类是对逻辑上相关的函数和数据的封装，它是对问题的抽象描述。相比函数，类的集成程度更高，也就更加适用于大型复杂程序的开发。

面向对象的程序设计方法要分析待解决的问题中包含哪些类事物，每类事物都有哪些特点，不同的事物种类之间是什么关系，事物之间如何相互作用等，这跟结构化程序设计考虑如何将问题分解成一个一个子问题的思路完全不同。

需要指出的是，面向对象的程序设计方法也离不开结构化的程序设计思想。编写一个类的内部代码细节时，还是要用结构化的设计方式。

5.1 类和对象的定义

5.1.1 类的声明

类是面向对象程序设计方法的核心，利用类可以实现对数据的封装和隐藏，任何类都是

数据成员和函数成员的封装体。与熟悉的 int、float 等基本类型不同，类是一种用户自定义类型，使用者根据需要可以自我定义，称为类的声明。类的声明的一般格式如下：

```
class 类名
{
private:
  私有成员变量和成员函数的声明
protected:
  保护成员变量和成员函数的声明
public:
  公有成员变量和成员函数的声明
};
```

其中 private、protected、public 分别表示对成员的不同访问权限的控制，它们的具体含义如下。

（1）private(私有)：私有成员，只能被类中的成员函数及该类的友元函数访问。

（2）protected(保护)：保护成员，只能被该类中的成员函数、派生类的成员函数或该类的友元函数访问。

（3）public(公有)：公有成员，可被与该类的对象处在同一作用域内的任何函数访问。

一般情况下，将成员变量声明为私有的，以便隐藏数据；而将部分成员函数声明为公有的，用于提供外界和这个类的对象的接口，从而使得其他函数(如 main()函数)可以访问和处理该类的对象。对于那些仅仅是为支持公有函数的实现而不作为对象接口的成员函数，也应该将它们说明为私有的。公有成员函数是外界所能观察到的对象接口，它们所表达的功能构成对象的功能，使同一个对象的功能能够在不同的软件系统中保持不变。这样，当数据结构发生变化时，只需要修改少量的代码(类的成员函数的实现代码)，就可以保持对象功能不变，只要对象的功能不变，则公有成员函数所定义的接口就不会发生改变。这样，对象内部实现所做的修改就不会影响使用该对象的软件系统。这就是面向对象程序设计使用数据封装为程序员开发活动带来的另外一个益处。

下面以三角形为例理解类的设计和定义过程。在面向对象的设计中，各种形状的三角形被称为对象。通过对这些对象进行分析，得到三角形对象都有三条边，三条边长度一旦确定，三角形的形状、周长和面积也就确定下来，由此设计出三角形类 Triangle 的定义。描述三角形对象属性的三条边的长度，在 C++中定义为该类的成员变量，计算三角形的面积和周长的操作，在 C++中定义为该类的成员函数。除此之外，还设计了三个成员函数，它们分别完成初始化三角形的三条边，测试三条边的长度是否满足组成三角形的要求和显示三角形的三条边的长度的操作。下面是三角形类 Triangle 的定义描述：

```
class Triangle
{
private:                                          //私有的成员变量和成员函数
    double a,b,c;                                 //三条边的长度
public:                                           //公有的成员变量和成员函数
    void setabc(double x, double y, double z)     //三角形初始化的函数
    {
        a = x;
```

```
        b = y;
        c = z;
    }
    void display()          //显示三条边的函数
    {
          cout << "a: " << a << "\t b: " << b << "\t c: " << c << endl;
    }
    bool isTriangle()       //测试是否为三角形的函数
     {
        return (a + b > c) && (a + c > b) && (b + c > a);
    }
     double area()          //求面积为函数
     {
        if (isTriangle())
         {
          double p = (a + b + c) / 2;
           return sqrt(p * (p - a) * (p - b) * (p - c));
         }
         else
            return -1;
    }
    double  peri()          //求周长为函数
     {
        if (isTriangle())
         {
          return a + b + c;
        }
        else
           return -1;
        }
};
```

类的定义有两种形式：类内定义和类外定义。前面的举例是类内定义形式，下面是类外定义形式。

```
class Triangle
{
private:
    double a,b,c;                           //三条边的长度
public:
    void setabc(int,int,int);               //三角形初始化的函数
    void display();                         //显示三条边的函数
    bool isTriangle();                      //测试是否为三角形的函数
    double area();                          //求面积的函数
    double peri();                          //求周长的函数
};
void Triangle::setabc(double x, double y, double z)
{
 a = x;
 b = y;
 c = z;
}
```

```
void Triangle::display()
{
 cout << "a: " << a << "\t b: " << b << "\t c: " << c << endl;
}
bool Triangle::isTriangle()
{
 return (a + b > c) && (a + c > b) && (b + c > a);
}
double Triangle::area()
{
 if (isTriangle())
 {
     double p = (a + b + c) / 2;
     return sqrt(p * (p - a) * (p - b) * (p - c));
 }
 else
     return -1;
}
double  Triangle::peri()
{
 if (isTriangle())
 {
     return a + b + c;
 }
 else
     return -1;
}
```

5.1.2 对象的定义

类的作用是定义对象。类和对象的关系如同一个模具与用这个模具铸造出来的零件之间的关系。类给出了属于该类的全体对象的抽象定义，而对象则是符合这种定义的特定的实体。所以，在C++中将对象称作类的一个实例。在程序中，每个对象需要自己的存储空间以保存它们自己的属性值，而所有对象共同使用类定义的操作代码。人们常说同类对象具有相同的属性和操作，是指它们的定义形式相同，而不是说每个对象的属性值都相同。

类的定义仅仅定义类的形式，它不占用任何内存来装载类的实例。只有定义类的对象，系统才会开辟相应的内存空间给对象，才能引用类的成员变量和成员函数。因此使用类之前，必须先定义类的一个实例，即对象。对象的本质就是变量，因此对象的定义也类似于变量的定义，而通常我们说对象是一个类的实例化。在定义一个对象前必须先定义好所属的类，一个类可以同时定义1个或多个对象，不同对象的成员变量的值是不同的。定义多个对象的方法如下：

类名　对象1，对象2，…，对象n；

例如，定义了Triangle类后，就可以通过它来创建对象了。

```
Triangle tri1, tri2, tri3;                    //定义 Triangle 类型的 3 个对象
```

5.1.3　对象成员的访问

当定义了类的对象之后，就可以通过对象成员访问运算符“.”来访问对象的公有成员，其一般形式如下：

对象名.成员;

例如：

```
tri1.setabc(3,4,5);          // setabc 已定义为公有成员函数
```

一般来说，对象的使用有以下 4 点规则。

(1) 使用“.”操作符访问其成员。

(2) 同类对象之间可以直接赋值，如 tri2＝tri1。

(3) 对象可以是函数的参数和返回值。

(4) 对象的成员也可以是对象。

【例 5-1】 定义三角形类的对象，并完成对象成员的访问。

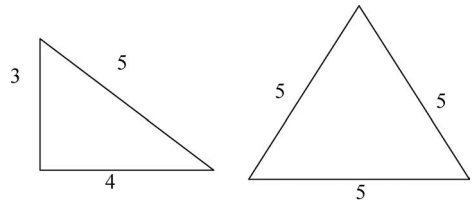
```
#include <iostream>
using namespace std;
class Triangle
{
private:
    double a,b,c;                          //三条边的长度
public:
void setabc(double x, double y, double z)   //三角形初始化的函数
{
    a = x;
    b = y;
    c = z;
}
void display()                             //显示三条边的函数
{
    cout << "a: " << a << "\t b: " << b << "\t c: " << c << endl;
}
bool isTriangle()                          //测试是否为三角形的函数
{
    return (a + b > c) && (a + c > b) && (b + c > a);
}
double area()                              //求面积的函数
{
    if (isTriangle())
    {
        double p = (a + b + c) / 2;
        return sqrt(p * (p - a) * (p - b) * (p - c));
    }
    else
        return -1;
}
double  peri()                             //求周长的函数
```

```
{
    if (isTriangle())
    {
        return a + b + c;
    }
    else
        return -1;
}
};
int main()
{
Triangle tri1,tri2;          //A
tri1.setabc(3,4,5);          //B
tri2.setabc(5,5,5);          //C
cout<<"tri1 的周长为:"<<tri1.peri()<<'\t'<<"面积为:"<<tri1.area()<<endl;
cout<<"tri2 的周长为:"<<tri2.peri()<<'\t'<<"面积为:"<<tri2.area()<<endl;
return 0;
}
```

程序的运行情况及结果如下：

```
tri1 的周长为:12    面积为:6
tri2 的周长为:15    面积为:10.8253
```

main()函数中 A 行定义了 Triangle 类的对象实例 tri1 和 tri2，B 和 C 行分别调用 setabc()函数设置了它们的三条边长，如图 5-1 所示。实际上，不同对象私有数据成员在内存中是互不相关的，共享的是成员函数定义。



(a) 三角形对象tri1　　(b) 三角形对象tri2

图 5-1　三角形的两个对象

5.2　构造函数和析构函数

构造函数和析构函数是类的两种特殊的成员函数。构造函数是在创建对象时，使用给定的值来将对象初始化。析构函数的功能正好相反，是在系统释放对象前，对对象做一些善后工作。

5.2.1　构造函数的定义

建立一个对象时，对象的状态/属性(成员变量的取值)是不确定的。为了使对象在创建的时候有确定的状态/属性，必须对其正确地初始化。构造函数作为特殊的成员函数，在定义对象时由系统自动调用，自动进行对象的初始化，因此类的构造函数一般定义为公有的。

类的构造函数定义一般形式如下：

```
类名([形参 1,形参 2,…,形参 n])
{
函数体
}
```

构造函数的函数体实现对私有数据成员的初始化功能，例如对于 Triangle 类，其构造函数定义如下：

```
Triangle(double x, double y, double z)                    //形式 1,对数据成员赋值
{
    a = x;
    b = y;
    c = z;
}
```

构造函数定义也可以使用初始化列表来实现对象的初始化。例如，对于以上的 Triangle，其构造函数还可以定义如下：

```
Triangle(double x, double y, double x): a(x), b(y), c(z)
 { }                  //形式 2 :使用初始化列表,函数体为空
```

构造函数的参数在排列时无顺序要求，只要保证相互对应即可。可以使用默认参数，也可以利用函数重载，定义多个构造函数。在程序中定义对象时，系统自动调用相应的构造函数来初始化对象。

在给类定义构造函数时，要注意以下 5 点。

(1) 构造函数的函数名与类名相同。

(2) 构造函数没有返回类型，返回类型也不能是 void。

(3) 构造函数可以有一个或多个参数，也可以没有参数，无参数的构造函数称为默认构造函数。

(4) 构造函数可以重载，即一个类可以有一个以上的构造函数。

(5) 同一个类的构造函数在重载时必须有不同的形参列表，列表在形参数量和类型上有所不同。

【例 5-2】 定义三角形类的构造函数，并完成对象成员的访问。

```
#include <iostream>
using namespace std;
class Triangle
{
    private:
      double a,b,c;                                          //三条边的长度
    public:
      Triangle(double x, double y, double z): a(x), b(y), c(z)   //A
      //使用初始化列表构造函数
     { }
    void display()                                           //显示三条边的函数
    {
        cout << "a: " << a << "\t b: " << b << "\t c: " << c << endl;
```

```
}
bool isTriangle()                          //测试是否为三角形的函数
{
    return (a + b > c) && (a + c > b) && (b + c > a);
}
double area()                              //求面积的函数
{
if (isTriangle())
{
    double p = (a + b + c) / 2;
    return sqrt(p * (p - a) * (p - b) * (p - c));
}
else
    return -1;
}
double  peri()                             //求周长的函数
{
if (isTriangle())
{
    return a + b + c;
}
else
    return -1;
}
};
int main()
{
Triangle tri1(3,4,5),tri2(5,5,5);     //B
cout <<"tri1 的周长为:"<< tri1.peri()<<'\t'<<"面积为:"<< tri1.area()<< endl;
cout <<"tri2 的周长为:"<< tri2.peri()<<'\t'<<"面积为:"<< tri2.area()<< endl;
return 0;
}
```

程序的运行情况及结果如下：

```
tri1 的周长为:12     面积为:6
tri2 的周长为:15     面积为:10.8253
```

A 行定义了构造函数，B 行在定义对象实例 tri1 和 tri2 同时调用 A 行构造函数设置了它们的边长。跟其他成员函数不同，构造函数在定义类的对象时由系统自动调用，而其他函数需要用函数名来显式调用。

5.2.2 构造函数的重载

在一个类中可以定义多个构造函数，以便提供对象不同的初始化的方法。这些构造函数具有相同的名字，而参数的个数或参数的类型不相同，这称为构造函数的重载。

【例 5-3】 构造函数的重载举例。

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

```
class Rect                                    //矩形类
{
    double a,b;
public:
    Rect(double a1 = 1)                       //默认值为 1 能创建正方形对象的构造函数
    {
        a = b = a1;
    }
    Rect(double a1, double b1):a(a1),b(b1)    //能创建矩形对象的构造函数
     { }
    double cir()                              //求周长的函数
    {
        return 2 * (a + b);
    }
    double area()                             //求面积的函数
    {
        return a * b;
    }
    void show()                               //输出矩形信息
    {
        cout <<"矩形边长分别为:"<< a <<'\t'<< b << endl;
    }
};
int main()
{
    Rect defaultrect;                         //调用默认值为 1 的能创建正方形对象的构造函数
    Rect rect1(5);                            //调用能创建正方形对象的构造函数
    Rect rect2(4,5);                          //调用能创建矩形对象的构造函数
    cout <<"defaultrect:";
    defaultrect.show();
    cout <<"面积 = "<< defaultrect.area()
        <<"\t 周长 = "<< defaultrect.cir()<< endl;
    cout <<"rect1:";
    rect1.show();
    cout <<"面积 = "<< rect1.area()<<"\t 周长 = "<< rect1.cir()<< endl;
    cout <<"rect2:";
    rect2.show();
    cout <<"面积 = "<< rect2.area()<<"\t 周长 = "<< rect2.cir()<< endl;
    return 0;
}
```

程序的运行情况及结果如下：

```
defaultrect:矩形边长分别为:1   1
面积 = 1    周长 = 4
rect1:矩形边长分别为:5 5
面积 = 25   周长 = 20
rect2:矩形边长分别为:4 5
面积 = 20   周长 = 18
```

注意：所有的对象在创建时，必须调用相应的构造函数，而且任一对象的构造函数必须唯一。

5.2.3 默认构造函数

默认构造函数(default constructor)有以下两种形式。

(1) 参数为默认值的构造函数。

如果在类体中声明以下形式的构造函数：

```
Triangle(double x = 5,double y = 5,double z = 5 );
```

用这个构造函数初始化对象时，可以提供全部参数、部分参数或不提供参数，对于没有提供的那部分参数，用默认值的参数值。例如，可以使用以下的方法定义对象：

```
Triangle tri1;                    //相当于 Triangle tri1(5,5,5);
Triangle tri2(3);                 //相当于 Triangle tri2(3,5,5);
Triangle tri3(3,4);               //相当于 Triangle tri3(3,4,5);
```

构造函数也可以部分是默认参数，其调用原则与一般的默认参数的函数一样，例如，在类体中说明以下形式的构造函数：

```
Triangle(double x,double y = 5,double z = 5);
```

用这个构造函数初始化对象时，因为形参 x 没有默认值，所以必须为 x 指定实参，因此，定义对象时至少要指定一个参数。

```
Triangle tri2(3);                 //相当于 Triangle tri2(3,5,5);
Triangle tri3(3,4);               //相当于 Triangle tri3(3,4,5);
Triangle tri4(3,4,4);             //相当于 Triangle tri4(3,4,4);
```

(2) 无参构造函数。

这种形式的构造函数可以显式定义，也可以由系统自动提供。无参构造函数的形式如下：

```
类名()
{
  函数体
}
```

如果类中没有定义任何构造函数时，C++编译器会自动提供一个默认构造函数，这个函数一般不执行任何操作，其函数体为空。例如，系统为 Triangle 类提供的默认构造函数形式如下：

```
Triangle()
{  }                              //函数体为空
```

用户显式定义的无参构造函数，函数体内可以有需要的语句。例如：

```
Triangle()
{
  a = b = c = 0;                  //将三角形的边长初始化为 0
}
```

由于是无参构造函数，定义类的对象时可以不提供参数。例如：

```
Triangle tri1;          //调用无参构造函数
```

默认构造函数的使用要注意以下 3 点。

（1）只要用户显式定义了一个类的构造函数，则编译系统就不再自动提供上面函数体为空的默认构造函数。

（2）参数全部默认的构造函数只能有一个。例如，类中不能同时出现以下两个构造函数的声明：

```
Triangle(double x = 5,double y = 5,double z = 5);
Triangle()
{
  a = b = c = 0;            //将三角形的边长初始化为 0
}
```

因为此时用语句“Triangle tri;”创建对象时，系统不能确定应该调用哪个构造函数。

（3）构造函数可以重载，但是每个对象调用的构造函数必须唯一。

【例 5-4】 默认构造函数举例。

```
#include <iostream>
using namespace std;
class Box                          //盒子类
{
private:
    int height,width,depth;        //长度,宽度,深度
public:
    Box();                         //无参构造函数
    Box(int,int,int);              //带 3 个参数的构造函数
    int volume();                  //容积函数
};
Box::Box()
{
    height = 1;
    width = 1;
    depth = 1;
}
Box::Box(int ht,int wd,int dp)
{
    height = ht;
    width = wd;
    depth = dp;
}
int Box::volume()
{
    return height * width * depth;
}
int main()
{
    Box thisbox(7,8,9);            //A 使用带参数的构造函数创建对象 thisbox
    Box defaultbox;                //B 使用默认不带参数的构造函数创建对象 defaultbox
```

```
    int volume = thisbox.volume();
    cout << volume << endl;
    int volume2 = defaultbox.volume();
    cout << volume2 << endl;
    return 0;
}
```

程序的运行情况及结果如下：

```
504
1
```

5.2.4 复制构造函数

复制构造函数是一种特殊的构造函数，具有一般构造函数的功能，它提供将一个已知对象的成员变量的值复制给正在创建的同类对象的方法，即使用已有的对象来复制一个新对象。

通常情况下，编译器建立一个默认的复制构造函数，复制构造函数采用复制方式使用已有的对象来创建新对象。

定义复制功能的构造函数的一般格式为：

```
类名(类名  &变量名)
{
函数体
}
```

复制构造函数的形参为引用变量，引用在类中一个很重要的用途就是定义复制构造函数。以前面的 Rect 类为例，系统默认的复制构造函数为：

```
Rect(Rect &r)          //Rect &r 表示定义 r 为 Rect 类的引用变量
{
a = r.a;
b = r.b;
}
```

【例 5-5】 系统默认的复制构造函数。

```
#include <iostream>
using namespace std;
class Book                              //书本类
{
    int length,width,pages;             //长度、宽度、页数
public:
    Book()                              //默认构造函数
    {
        length = 260;
        width = 185;
        pages = 365;
    }
    Book(int len,int wid,int pag)       //有参构造函数
```

```
    {
        length = len;
        width = wid;
        pages = pag;
    }
    void display()
    {
        cout <<"Length: "<< length <<"\t Width: "<< width << endl;
    }
};
int main()
{
    Book defaultbook;                    //调用默认构造函数定义对象
    Book copybook1(defaultbook);         //调用系统默认的复制构造函数
    defaultbook.display();
    copybook1.display();
    Book thisbook(184,130,265);          //调用有参构造函数
    Book copybook2(thisbook);            //调用系统默认的复制构造函数
    thisbook.display();
    copybook2.display();
    return 0;
}
```

程序的运行情况及结果如下：

```
Length: 260      Width: 185
Length: 260      Width: 185
Length: 184      Width: 130
Length: 184      Width: 130
```

5.2.5　析构函数

构造函数的作用是定义对象时分配内存空间，使用给定的值初始化对象。析构函数(destructor)的作用恰恰相反，在释放对象之前做一些必要的清理工作，主要是清理系统分配的对象内存，它们的调用都不需要用户干涉，当创建一个对象时，系统自动调用该类的构造函数，当对象退出作用域时，系统自动调用该类的析构函数。析构函数也与类同名，为了与构造函数区分，在析构函数的前面加上一个“～”符号。一个类只有一个析构函数，析构函数的类外声明如下：

```
～类名()
{
函数体
}
```

以前面的类 Book 为例，系统默认的析构函数如下：

```
～Book()
{ }
```

析构函数的特点如下。

（1）析构函数是一个特殊的成员函数，函数名必须与类名相同，并在其前面加上符号“～”，以便和构造函数名区别。

（2）析构函数不能带有任何参数，不能有返回值，不指定函数类型。

（3）一个类中，只能定义一个析构函数，析构函数不允许重载。

（4）析构函数是在撤销对象时由系统自动调用的。

在程序的执行过程中，当遇到某一对象的生存期结束时，系统自动调用析构函数，然后再收回为对象分配的存储空间。

不同存储类型的对象调用构造函数和析构函数的顺序如下。

（1）对于全局定义的对象（在函数外定义的对象），在程序开始执行时，调用构造函数；到程序结束时，调用析构函数。

（2）对于局部定义的对象（在函数内定义的对象），当程序执行到定义对象的地方时，调用构造函数，在退出对象的作用域时，调用析构函数。

（3）用 static 定义的局部对象，在首次到达对象的定义时调用构造函数，到程序结束时，调用析构函数。

【例 5-6】 析构函数的应用。

```
#include<iostream>
#include<cstring>
using namespace std;
class  A
{
   float   x,y;
public:
   A(float a, float b)
   {
     x=a;
     y=b;
     cout<<"初始化自动局部对象 x="<<x<<" y="<<y<<endl;
   }
   A()
   {
    x=0;
    y=0;
    cout<<"初始化静态局部对象 x="<<x<<" y="<<y<<endl;
}
 A(float  a)
{
    x=a;
    y=0;
    cout<<"初始化全局对象 x="<<x<<" y="<<y<<endl;
}
 ～A()
{
  cout<<"调用析构函数 x="<<x<<" y="<<y<<endl;
}
};
A  a0(100.0);                    //定义全局对象
void f()
```

```
{
  cout<<" -->进入 f()函数\n";
   A  ab(10.0,  20.0);          //定义局部自动对象
   static  A  a3;               //初始化局部静态对象
}
int main()
{
  cout<<"进入 main()函数\n ";
  f();
  f();
  return 0;
}
```

程序的运行情况及结果如下：

```
初始化全局对象 x=100 y=0
进入 main()函数
  -->进入 f()函数
初始化自动局部对象 x=10 y=20
初始化静态局部对象 x=0 y=0
调用析构函数 x=10 y=20
-->进入 f()函数
初始化自动局部对象 x=10 y=20
调用析构函数 x=10 y=20
调用析构函数 x=0 y=0
调用析构函数 x=100 y=0
```

5.3　静态成员

5.3.1　静态成员变量

通常情况下，每次创建一个对象时，编译系统把该类中的有关成员变量复制到该对象中，即同一类的不同对象，其成员变量之间是互相独立的、独立存储的。

当我们将类的某一个成员变量的存储类型指定为静态类型时，则该类所产生的所有对象，其静态成员均共享一个存储空间，这个空间是在编译的时候分配的。换言之，在说明对象时，不再为静态类型的成员额外分配空间。

在类定义中，用关键字 static 修饰的成员变量称为静态成员。

有关静态成员变量的使用，说明以下 4 点。

(1) 类的静态成员变量是静态分配存储空间的，而其他成员是动态分配存储空间的(全局变量除外)。当类中没有定义静态成员变量时，在程序执行期间遇到说明类的对象时，才为对象的所有成员依次分配存储空间，这种存储空间的分配是动态的。而当类中定义了静态成员变量时，在编译时，就要为类的静态成员变量分配存储空间。

(2) 必须在文件作用域中，对静态成员变量做一次且只能做一次定义性声明。因为静态成员变量在定义性声明时已分配了存储空间，所以通过静态成员变量名前加上类名和作

用域运算符，可直接引用静态成员变量。在 C++中，静态变量缺省的初值为 0，所以静态成员变量总有唯一的初值。当然，在对静态成员变量做定义性的声明时，也可以指定一个初值。

（3）静态成员变量兼具有全局变量的生命期和成员变量的访问权限的特性。静态成员变量与全局变量一样都是静态分配存储空间的，但全局变量在程序中的任何位置都可以访问它，而静态成员变量受到访问权限的约束。必须是 public 权限时，才可能在类外进行访问。

（4）为了保持静态成员变量取值的一致性，通常在构造函数中不给静态成员变量置初值，而是在对静态成员变量的定义性声明时指定初值。

例如：

```
class Cuboid                    //立方体类
{
public:
    //...
    static  int  count;         //使用静态成员变量,表示立方体的总数量
};
```

应用程序中声明对象如下：

```
Cuboid cub1, cub2, cub3;
```

则对象 cub1、cub2、cub3 共享静态成员变量 count。

由于静态成员变量是在类的范畴，对于该类所有的对象共享的存储单元，存储在静态数据区，因此静态成员变量必须在应用之前初始化。

【例 5-7】 静态成员变量的应用。

```
#include <iostream>
using namespace std;
class  Cuboid                               //立方体类
{
    private:
        int  a;                             //棱长
        static  int count;                  //静态成员变量的声明,count 表示对象个数
    public:
        Cuboid(int a1 = 1)                  //构造函数
        {
           a = a1;
            count++;
            cout <<"Number of Cuboids = "<< count <<'\n';
        }
        ~ Cuboid()                          //析构函数
        {
           count -- ;
            cout <<"Number of Cuboids = "<< count <<'\n';
        }
        void show()                         //显示函数
        {
```

```
            cout << " Cuboid a = "<< a <<'\n';
             cout << "count = "<< count <<"\n";
        }
};
int Cuboid::count = 0;                    //静态成员变量的初始化
int  main()
{
     Cuboid  cub1(20),cub2,cub3;          //定义三个对象
    cub1.show();
    return 0;
}
```

程序的运行情况及结果如下：

```
Number of Cuboids = 1
Number of Cuboids = 2
Number of Cuboids = 3
Cuboid a = 20
count = 3
Number of Cuboids = 2
Number of Cuboids = 1
Number of Cuboids = 0
```

静态数据成员用得比较多的场合一般为以下两种。

(1) 用来保存流动变化的对象个数(如例 5-7 的 count)。

(2) 作为一个标识,指示一个特定的动作是否发生(如可能创建几个对象,每个对象要对某个磁盘文件进行写操作,但显然在同一时间里只允许一个对象写文件,在这种情况下,用户希望利用一个静态成员变量,指出文件何时正在使用,何时处于空闲状态)。

5.3.2　静态成员函数

例 5-7 中的静态数据成员是通过普通成员 show()函数显示出来的,这种使用并不规范。因为静态数据成员是属于类的,不是属于哪个对象的。为了方便访问静态数据成员,使用静态成员函数访问静态数据成员。静态成员函数与静态数据成员一样是属于类的,它们不是任何对象的组成部分。

对静态成员函数的用法说明以下 5 点。

(1) 与静态成员变量一样,在类外的程序代码中,通过类名加上作用域操作符::,可直接调用静态成员函数。

(2) 静态成员函数只能直接调用该类的静态成员变量或静态成员函数,不能直接调用非静态的成员变量。这是因为静态成员函数可被其他程序代码直接调用,所以,它不包含对象地址的 this 指针。

(3) 静态成员函数的实现部分在类定义之外定义时,其前面不能加修饰词 static。这是由于关键字 static 不是数据类型的组成部分,因此,在类外定义静态成员函数的实现部分时,不能使用这个关键字。

（4）不能把静态成员函数定义为虚函数。静态成员函数也是在编译时分配存储空间的，所以在程序的执行过程中不能提供多态性。

（5）可将静态成员函数定义为内联函数(inline)，其定义方法与非静态成员函数完全相同。

【例 5-8】 静态成员函数。

```
#include<iostream>
using namespace std;
class  Cuboid                                   //立方体类
{
private:
    int a;                                      //棱长
    static int count;                           //静态成员变量的声明,count 表示对象个数
public:
    Cuboid(int a1 = 1)
    {
        a = a1;
        count++;
    }
    ~Cuboid()
    {
        count--;
    }

    void show()                                 //显示函数
    {
        cout << " Cuboid a = "<< a <<'\n';
        cout << "count = "<< count <<"\n";
    }
    static int TotalNumber()                    //静态成员函数,表示返回对象个数
     {
         return count;
    }
};
int Cuboid::count = 0;                          //静态成员变量的初始化
int  main()
{   Cuboid  cub1(20),cub2,cub3;
    cout << Cuboid::TotalNumber()<< endl;       //用类名引用静态成员函数
    cout << cub1.TotalNumber()<< endl;          //用对象引用静态成员函数
    return 0;
}
```

程序的运行情况及结果如下：

```
3
3
```

从运行结果可以看到，既可以用类名引用静态成员函数，也可以用该类的对象引用静态成员函数，两者结果一样。

5.4　常成员

在程序设计中，如果既想要数据能在一定范围内被共享，又要保证它不被任意修改，可以将该数据用 const 修饰为常量，因为常量在程序运行期间是不可改变的。在类的定义中常量中有类的常成员，常成员包括常成员变量和常成员函数。

5.4.1　常成员变量

在声明类的成员变量时，前面加上 const 关键字，表示该成员变量初始化之后不能再改变。注意，常成员变量的唯一初始化方法，就是用构造函数的初始化列表完成初始化。通常把常成员变量定义为静态成员，使其成为类的一个常量。

【例 5-9】 常成员变量的应用。

```
#include<iostream>
using namespace std;
class Circle                                    //圆类
{
private:
    int r;                                      //圆半径
    const int id;                               //圆 id
    static const double PI;                     //静态常成员变量 ,PI 表示圆周率
public:
    Circle(int i, int r1);                      //有参构造函数
    void show();                                //显示函数
    double area();                              //面积函数
    double cir();                               //周长函数
};
//静态常成员变量,类外初始化
const double Circle::PI = 3.14159;
//常成员变量只能通过初始化列表,获得初始值
//id 为常成员变量,不能把 id=i 写到构造函数体内,必须通过初始化列表实现初始化
//普通成员 r 也可在初始化列表中赋值
Circle::Circle(int i, int r1) :id(i), r(r1){ }   //通过初始化列表实现初始化
double Circle::cir()
{
    return 2 * PI*r;
}
double Circle::area()
{
    return PI*r*r;
}

void Circle::show()
{
    cout << "Circle id = " << id << "\t r = " << r << endl;
}
int main()
{
```

```
    Circle circle1(20, 1);
    circle1.show();
    cout << circle1.area() << endl;
    return 0;
}
```

程序的运行情况及结果如下：

```
Circle id = 20   r = 1
3.14159
```

5.4.2 常成员函数

const 成员函数可以访问所有的成员变量，但是不能修改它们的值，const 关键字的作用有两点：一是限制不能修改传入参数的值，二是提醒程序员这是 const 函数，不要修改。

常成员函数的定义需要注意的地方有两点。

（1）const 关键字需要加在函数声明的尾部，因为加在头部的表示返回值是 const 变量。

（2）在函数声明和函数定义的函数名尾部都要加 const 关键字。

【例 5-10】 常成员函数的应用。

```
#include<iostream>
using namespace std;
class Circle                                    //圆类
{
private:
    int r;                                      //圆半径
    const int id;                               //常成员变量,id表示圆 id
    static const double PI;                     //静态常成员变量,PI表示圆周率
public:
   Circle(int i, int r1);
   void show();
   int getId() const                            //常成员函数,返回常成员
  {
      return id;
  }
    const double getPI() const                  //常成员函数,返回静态常成员
  {
      return PI;
  }
};
//静态常成员变量,类外初始化
const double Circle::PI = 3.14159;
//常成员变量只能通过初始化列表,获得初始值
Circle::Circle(int i, int r1):id(i),r(r1){ }
 void Circle::show()
{
   cout << id << endl;
}
int main()
```

```
{
    Circle c1(20,1);
    c1.show();
    cout << c1.getId()<< endl;
    cout << c1.getPI()<< endl;
    return 0;
}
```

程序的运行情况及结果如下：

```
20
20
3.14159
```

5.5　结构体

5.5.1　结构体类型

前面我们讨论了C++语言所提供的各种基本数据类型，例如int、short int、long int、float、double、long double等。这些基本数据类型对于描述复杂应用问题中的多种类型的数据是远远不够的。程序设计语言一般都提供了便于程序员定义自己所需要的数据类型的机制，这就是自定义数据类型，从而大幅提高了程序设计语言描述复杂数据对象的能力。

结构体是C语言中的自定义类型，在C++中继续保留。结构体的引入给C语言中的逻辑处理带来了更多的方便，如描述一个人的数据信息，包括姓名、年龄、体重、性别等，这些数据信息的类型和含义是不一样的。结构体允许用户定义一种新的数据类型，把属于同一个事物的若干相关数据构成一个整体，统一管理，这种新的数据类型称为结构体类型。在C++中，结构体与类类似，但有着本质的区别。

结构体声明的语法格式如下：

```
struct  结构体类型名
{
  数据类型  成员变量1;
  数据类型  成员变量2;
     ⋮
  数据类型  成员变量n;
};
```

其中struct是结构体类型声明的关键字，结构体类型名必须是合法的标识符，成员变量1，成员变量2，…，成员变量n是互不同名的成员项，表示数据集中包括的各项数据。例如，我们可以用一个结构体来描述一个长方形类型square_type。

```
struct square_type
{
     double a;              //长
    double b;               //宽
};
```

于是，square_type 就是一个已经声明了的程序员可以使用的结构体数据类型，接下来就可以定义 square_type 类型变量了。

5.5.2 结构体变量

定义一个类型为 square_type 的结构体变量 square 的方法如下：

```
square_type square;
```

一个结构体变量所分配到的是一块连续的内存空间，各个成员在这块空间中依次顺序存储，这块内存空间的字节数是它的所有成员各自所需的内存字节数的总和，例如，我们刚才定义的结构体变量 square 占有 16 字节的内存空间。

结构体变量先声明结构体类型，后定义变量，再使用。例如：

```
struct date_type                        //日期结构体类型
{
   unsigned short year;                 //年
   unsigned short month;                //月
   unsigned short day;                  //日
};
data_type d1,d2;                        //定义两个日期类型变量 d1,d2
```

先声明了一个用来描述日期类型的结构体类型 date_type，然后用 date_type 定义了两个表示不同日期的结构体变量 d1 和 d2。又例如：

```
struct time_type                        //时间结构体类型
{
   unsigned short hour;                 //时
   unsigned short minute;               //分
   unsigned short second;               //秒
};
time_type t1,t2;                        //定义两个时间类型变量 t1,t2
```

先声明了一个用来描述时间类型的结构体类型 time_type，然后用 time_type 定义了两个表示不同时间的结构体变量 t1 和 t2。

与其他类型变量的初始化一样，对结构体变量的初始化，可以在结构体变量定义时指定其初始值。例如：

```
date_type d = {2022,12,8};
```

定义了一个表示日期的结构体变量 d，并且给结构体变量 d 的各个成员 d. year、d. month、d. day 指定了初始值，即 d. year 的值是 2022，d. month 的值是 12，而 d. day 的值是 8。

由于结构体变量是多种数据类型的组合，所以不能以变量的形式整体进行赋值或运算，例如：

```
date_type d;
d = {2022,12,8};                        //错误
```

对结构体变量的操作是通过对该变量的各个成员的操作来实现的，引用结构体变量的

成员的形式如下：

结构体变量名.成员

其中，“.”是成员运算符，在所有的C++运算符中优先级最高。

结构体变量在内存中依照其成员的顺序排列，所占内存空间的大小是其全体成员所占空间的总和。在编译时，编译系统仅对结构体变量分配空间，不对结构体类型分配空间。对结构体中各个成员可以单独引用、赋值，其作用与变量等同。结构体的成员可以是另一个结构体类型的变量。

关于结构类型变量的使用，说明以下3点。

（1）同类型的结构体变量之间可以直接赋值。这种赋值等同于各个对应成员的依次赋值。

（2）结构体变量不能直接进行输入输出，它的每一个成员能否直接进行输入输出，取决于其成员的类型，若是基本类型或是字符串，则可以直接输入输出。

（3）结构体变量可以作为函数的参数，函数也可以返回结构体类型的值。当函数的形参与实参为结构体类型的变量时，这种结合方式属于值传递（传值调用）。

【例5-11】 结构体类型的应用。

表5-1是一个应用问题中要求程序组织和处理的书籍基本情况，在这里，一个数据元素是一本书籍的所有信息的集合，包含书本的ISBN、书名、出版社、出版时间和价格。显然，这些属性从不同的角度刻画了一本书籍的不同状态或者特征。一般来说，它们具有不同的数据类型。对于表5-1来说，可以用一个结构体类型来描述。

表5-1　书籍基本情况

ISBN	书　　名	出版社	出版时间	价格
7302652090	C++从入门到精通（第6版）	清华大学出版社	2024.6.1	99.8
7302627739	深入浅出数据结构与算法（微课视频版）	清华大学出版社	2023.4.1	99
7302644156	启发式优化算法理论及应用	清华大学出版社	2023.10.1	59
7302635284	Java项目驱动开发教程	清华大学出版社	2023.6.1	89
7302649717	高效C/C++调试	清华大学出版社	2024.1.1	99

```
#include <iostream>
#include <string>
using namespace std;
struct time                          //出版时间结构体类型
{
  unsigned short year;
  unsigned short month;
};
struct Book                          //书本结构体类型
{
  string serial_number;              //ISBN
  string name;                       //name为string对象，书名
  string publishing_house;           //publishing_house为string对象，出版社
  time  publishing_date;             //该结构体成员是另一个结构体类型的变量，出版时间
  float price;                       //价格
};
```

```
int main()
{
    Book book1;
    book1.serial_number = "7302652090";
    book1.name = "C++从入门到精通(第 6 版)";
    book1.publishing_house = "清华大学出版社";
    book1.publishing_date.year = 2024;
    book1.publishing_date.month = 6;
    book1.price = 99.8;
    //输出数据
    cout << book1.serial_number <<'\n'
         << book1.name <<'\n'
         << book1.publishing_house <<'\n'
         << book1.publishing_date.year <<' - '
         << book1.publishing_date.month <<'\n'
        <<' ¥ '<< book1.price << endl;
    return 0;
}
```

程序的运行情况及结果如下：

```
7302652090
C++从入门到精通(第 6 版)
清华大学出版社
2024 - 6
 ¥ 99.8
```

5.6 枚举

如果用变量 tomorrow 表示明天的序号，那么我们对 tomorrow 可以有两种理解，一是它表示明天是几号，这时可以用语句 int tomorrow，把 tomorrow 定义为 int 型的变量；二是它表示明天是星期几，这时也可以用语句 int tomorrow，把 tomorrow 也定义为 int 型的变量。我们看到，对于 tomorrow 的两种不同的理解，上面却用了相同的 C++的描述形式。更为重要的是，按照第一种理解，tomorrow 的取值范围是 1 到 31，按照第二种理解，tomorrow 的取值范围是 1 到 7，然而我们都用 int 型来定义 tomorrow。数据类型 int 所规定的取值范围远远超过了我们对 tomorrow 的两种不同的理解的取值范围，即一个变量的实际取值范围与它的数据类型所规定的数据范围不一致。C++语言提供的枚举类型的定义机制有助于程序员避免这种不一致，另外，枚举类型机制也有利于提高程序的可读性。

当一个变量只能取给定的几个值时，则可以定义其为枚举类型。枚举类型声明的语法如下：

```
enum  枚举类型名
{
     枚举常量 1,枚举常量 2,…,枚举常量 n
};
```

例如，为了描述一周之内的某一天，我们可以用保留字 enum 声明一个枚举类型

weekday_type：

```
enum weekday_type                    //星期枚举类型
{
    SUNDAY, MONDAY, TUESDAY, WEDNESDAY, THURSDAY, FRIDAY, SATURDAY
};
```

其中，SUNDAY、MONDAY、TUESDAY、WEDNESDAY、THURSDAY、FRIDAY 和 SATURDAY 是枚举类型 weekday_type 的所有枚举常量，用以表明以 weekday_type 为类型的变量的取值只能是这 7 个枚举常量中的某一个。另外，C++语言的编译程序将根据源程序中枚举常量的书写顺序，为每个枚举常量都规定一个内部值。枚举类型中，第一个枚举常量的内部值是 0；如果一个枚举常量的内部值是 i，则它后面的那个枚举常量的内部值是 i+1。在一定的作用域内，一个枚举常量本身与它的内部值是等价的。例如，在枚举类型 weekday_type 中 SUNDAY 的内部值是 0，MONDAY 的内部值是 1，以此类推。现在就可以用枚举类型 weekday_type 来定义所需要的枚举变量了。

```
weekday_type  today, tomorrow;
if (today == SUNDAY)
        tomorrow = MONDAY;
```

这里的 if 语句反映了枚举变量 today 和 tomorrow 之间的取值关系。实际上，根据枚举常量内部值的规定，枚举变量 today 和 tomorrow 之间的取值关系可以由下面的语句说明：

```
tomorrow = (today + 1) % 7;
```

C++语言还允许程序员指定枚举常量的内部值。例如：

```
enum weekday_type
{SUNDAY = 7, MONDAY = 1, TUESDAY, WEDNESDAY, THURSDAY, FRIDAY, SATURDAY};
```

则 SUNDAY 的内部值是 7，MONDAY 的内部值是 1，TUESDAY 的内部值是 2，以此类推。

【例 5-12】 枚举类型的应用。

有一张表格，它依次记录了某国从 1 月到 12 月实际每个月的产值 GDP，要计算某国全年的总产值 GDP。这个问题在学习一维数组的时候就可以解决了。现在我们只是就这个问题而言，在下面的程序中月份采用枚举类型，函数中手工输入每月的产值，将返回某国的全年的总产值 GDP。

```
#include <iostream>
using namespace std;
enum month
{Jan = 1,Feb,Mar,Apr,May,Jun,Jul,Aug,Sep,Oct,Nov,Dec};
int main()
{
  float yearearn,monthearn;
  month m;
  yearearn = 0;
```

```
    for(m = Jan; m <= Dec; m = month(m + 1))
    {
      cout <<"Enter the monthly earning for ";
      switch(m)
      {
        case Jan   : cout <<"January.     \n"; break;
        case Feb   : cout <<"February.    \n"; break;
        case Mar   : cout <<"March.       \n"; break;
        case Apr   : cout <<"April.       \n"; break;
        case May   : cout <<"May.         \n"; break;
        case Jun   : cout <<"June.        \n"; break;
        case Jul   : cout <<"July.        \n"; break;
        case Aug   : cout <<"August.      \n"; break;
        case Sep   : cout <<"September.   \n"; break;
        case Oct   : cout <<"October.     \n"; break;
        case Nov   : cout <<"November.    \n"; break;
        case Dec   : cout <<"December.     \n"; break;
      }
      cin >> monthearn;
      yearearn += monthearn;
    }
    cout <<"The GDP of the country:"<< yearearn << endl;
    return 0;
}
```

程序的运行情况及结果如下：

```
Enter the monthly earning for January.
3↙
Enter the monthly earning for February.
1↙
Enter the monthly earning for March.
4↙
Enter the monthly earning for April.
5↙
Enter the monthly earning for May.
7↙
Enter the monthly earning for June.
9↙
Enter the monthly earning for July.
10↙
Enter the monthly earning for August.
12↙
Enter the monthly earning for September.
3↙
Enter the monthly earning for October.
3↙
Enter the monthly earning for November.
4↙
Enter the monthly earning for December.
7↙
The GDP of the country:68
```

5.7　综合举例

综上所述，面向对象的编程可以分为以下两个步骤。

（1）确定类的功能，实际上就是定义一个类，根据类要实现的功能确定类的成员数据和成员函数。

（2）编写 main()函数，验证类的各个功能的正确性。

【例 5-13】 利用面向对象的编程方法求两个数的最大公约数和最小公倍数。

定义一个类 Num，实现求两个数的最大公约数和最小公倍数的功能，类中包括：

（1）私有成员数据。

int x,y：存放两个整数。

（2）公有成员函数。

Num(int a,int b)：构造函数，用于初始化私有数据成员。

int gys(x,y)：利用欧几里得算法求 x 和 y 的最大公约数，作为函数值返回。

int gbs(x,y)：求 x 和 y 的最小公倍数，作为函数值返回。

```
#include<iostream>
using namespace std;
class Num                          //类名,求两数的最大公约数和最小公倍数
{
 int x,y;                          //私有数据
public:
 void Num(int a, int b);           //构造函数
 int gys();                        //求最大公约数
 int gbs();                        //求最小公倍数
};
Num::Num(int a, int b)
{
  x=a;
  y=b;
}
int Num::gys()                     //用欧几里得算法求 m、n 的最大公约数
{
   int r, m,n;
   m=x;
   n=y;
   if(m<n)                         //要求 m 大于 n,当 m 小于 n 时,交换 m、n 的值
   {
         r=m;
         m=n;
         n=r;
    }
    while(r=m%n)                   //r 不为 0,循环迭代
    {
      m=n;
      n=r;
    }
   return n;                       //返回最大公约数的值
```

```
}
int Num::gbs()
{
    int r = gys();
    return x * y/r;                    //两数的最小公倍数是两数之积除以最大公约数
}
int main()
{
    int a,b;
    cout <<"请输入两个整数: ";
    cin >> a >> b;
    Num num(a,b);                      //定义类的对象 num,并调用构造函数初始化对象
    cout << a <<" , "<< b <<"的最大公约数是: "<< num.gys()<<'\t';
    cout <<"最小公倍数是: "<< num.gbs()<< endl;
    return 0;
}
```

程序的运行情况及结果如下：

```
请输入两个正整数:12 16↙
12  ,  16 的最大公约数是:4          最小公倍数是:48
```

【例 5-14】 体育场改造预算。

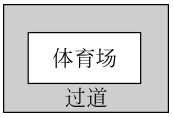


图 5-2 矩形体育场平面图

某矩形体育场如图 5-2 所示，现在需在其周围建一矩形过道，并在四周围上栅栏。栅栏单价为 50 元/米，过道造价单价为 300 元/米^2，过道宽为 3 米，体育场的长宽由键盘输入。请编写程序计算并输出过道和栅栏的总造价。

分析：过道总造价＝过道面积×造价单价，过道面积＝外体育场面积－内体育场面积。这里的内体育场是指真正的体育场，外体育场就是内体育场加上一圈过道得到的体育场。栅栏总造价＝栅栏长度×单价，栅栏长度＝内体育场周长。建立矩形类和体育场类，矩形类属性为长和宽，成员函数为周长函数和面积函数。体育场类设计为成员数据，有内外两个体育场对象，而栅栏单价、过道造价单价和长度相对稳定，就设置为静态常成员，普通成员函数为总造价函数、栅栏总造价函数和过道总造价函数。

```
#include <iostream>
using namespace  std;
class Rect                         //长方形类
{
    private:
        int a,b;                   //长,宽
    public:
        Rect(int a1,int b1):a(a1),b(b1){ }
        int c()                    //周长函数
        {
            return 2 * (a + b);
        }
        int s()                    //面积函数
```

```
        {
            return a * b;
        }
};
class Stadium                    //体育场类
{
    private:
        Rect irect,orect;        //内长方形,外长方形
        //静态常成员变量 cca 表示过道造价,ccf 表示栅栏造价, lena 表示过道长度
        static const int cca,ccf,lena;
    public:
        //构造函数参数的 a1 和 b1 是指真正的体育场长和宽,也就是内体育场长和宽
        Stadium(int a1,int b1):irect(a1,b1),orect(a1 + 2 * lena,b1 + 2 * lena){ }

        int cc()                 //总造价函数
        {
            return  (orect.s() - irect.s()) * cca + ccf * irect.c();
        }
        int ccas()               //过道总造价函数
        {
            return (orect.s() - irect.s()) * cca;
        }
        int ccfs()               //栅栏总造价函数
        {
            return  ccf * irect.c();
        }
};
const int Stadium::cca = 300;
const int Stadium::ccf = 50;
const int Stadium::lena = 3;
int main()
{
    int a,b,p;
    cout <<"请输入体育场总长度和总宽度:";
    cin >> a >> b;
    Stadium s = Stadium(a,b);
    p = s.cc();
    cout <<"该体育场工程总造价为 "<< p <<"元 \n"
        <<"其中过道总造价为 "<< s.ccas()<<"元 \n"
        <<"栅栏总造价为 "<< s.ccfs()<<"元"<< endl;
    return 0;
}
```

程序的运行情况及结果如下：

```
请输入体育场总长度和总宽度:100 50↙
该体育场工程总造价为 295800 元
其中过道总造价为 280800 元
栅栏总造价为 15000 元
```

练习题

一、选择题

1. 有以下类定义：

```
class MyClass
{
   char a;
   int b;
   double c;
public:
  MyClass():c(0.0),b(0),a(','){}
};
```

创建这个类的对象时，数据成员的初始化顺序是_____。

A. a,b,c　　B. c,b,a　　C. b,a,c　　D. c,a,b

2. 下面选项中，对类 A 的析构函数的正确定义是_____。

A. ～A::A()　　B. void ～A::A(参数)

C. ～A::A(参数)　　D. void ～A::A()

3. 下面有关构造函数的不正确说法是_____。

A. 构造函数可以用来实现所有成员变量的初始化

B. 构造函数不是类的成员函数

C. 当生成类的实例时，自动调用构造函数进行初始化

D. 构造函数用来分配对象所需的内存

4. 下面有关类的说法错误的是_____。

A. 一个类可以有多个构造函数　　B. 一个类只能有一个析构函数

C. 可以给析构函数指定参数　　D. 一个类中可以说明具有类型的成员变量

5. 以下程序的运行结果是_____。

```
#include <iostream>
using  namespace std;
class Test
{
public:
   Test(){}
   Test(Test * t){cout << 1;}
 };
Test fun(Test &u)
{
   Test t = u;
   return t;
}
int main()
{
Tes x,y;
```

```
x = fun(y);
return 0;
}
```

A. 无输出　　B. 1　　C. 11　　D. 111

6. 以下程序的运行结果是_____。

```
#include < iostream >
using namespace std;
class Con
{
char ID;
public:
    char getID()
    {return ID;}
    Con(){ID = 'A';cout << 1;}
    Con(char id){ID = id;cout << 2;}
    Con(Con &c){ID = c.getID();cout << 3;}

};
void show(Con c)
{
cout << c.getID();
}
int main()
{
Con c1;
show(c1);
Con c2('B');
show(c2);
return 0;
}
```

A. 1A2B　　B. 13A23B　　C. 13A2B　　D. 1A23B

二、填空题

1. 阅读以下一段程序，请写出程序的输出结果_____。

```
#include < iostream >
using  namespace std;
class base
 {
     private:
        int x;
     public:
       void setX(int a){x = a;}
        int getX(){return x;}
};
int main()
{
base a;
a.setX(55);
```

```
cout << a.getX()<< endl;
return 0;
}
```

2. 阅读以下一段程序，请写出程序的输出结果_____。

```
#include <iostream>
#include <iomanip>
using  namespace std;
//Time abstract data type(ADT) definition
class Time
{
public:
    Time();                       //constructor
    void setTime(int,int,int);    //set hour,minute,second
    void printUniversal();        //print universal time format
    void printStandard();         //print standard time format
private:
    int hour;                     //0 - 23(24 - hour clock format)
    int minute;                   //0 - 59
    int second;                   //0 - 59
};                                //end class Time
Time::Time()
{
hour = minute = second = 0;
}
void Time::setTime(int h, int m, int s)
{
   hour = (h >= 0 && h < 24) ? h:0;
   minute = (m >= 0 && m < 60) ? m:0;
   second = (s >= 0 && s < 60) ? s:0;
}
void Time::printUniversal()
{
   cout << setfill('0')<< setw(2)<< hour <<":"
        << setw(2)<< minute <<":"
        << setw(2)<< second;
}
void Time::printStandard()
{
   cout <<((hour == 0||hour == 12)?12:hour % 12)
        <<":"<< setfill('0')<< setw(2)<< minute
        <<":"<< setw(2)<< second
        <<(hour < 12?"AM":"PM");
}
int main()
{
    Time t1;
    t1.setTime(18,22,9);
    cout <<"this time is:";
    t1.printStandard();
}
```

三、编程题

1. 设计一个复数类，包含表示实数部分和虚数部分的成员数据，内有成员函数来计算两个复数的加、减、乘、除。编写完整程序测试复数类。

2. 设计一个点类，包含表示横、纵坐标值的成员数据，内有成员函数来计算与另一个点的距离。编写完整程序测试点类。

3. 设计一名学生结构体，包含学号、姓名、性别、生日、三门课成绩（即数学、语文、英语）。编写完整程序求每名学生的总成绩。