第2章：类和对象之初体验

计算机系 江志英 jiangzy@mail.buct.edu.cn

本章内容

5.1 类的定义

5.2 对象的定义

5.3 对象的初始化

5.4 成员函数的特性

5.5 静态成员

5.6 友元

计算机系 江志英 jiangzy@mail.buct.edu.cn

1 类的定义

- **1.1** 什么是类
- **1.2** 类的定义格式
- **1.3** 注意事项

计算机系 江志英 **jiangzy**@mail.buct.edu.cn

什么是类

- 类是一种复杂的数据类型，它将不同类型的数据和与这些数据相关的操作封装在一起的集合体。
- 类是一种用户自定义类型
- 类的结构是用来确定一类对象的行为的，而这些行为是通过类的内部数据结构和相关的操作来确定的。
- 类是面向对象程序设计方法的核心。在面向对象程序设计中，程序模块是由类构成的。

类的定义格式

```
//说明部分
    class <类名>
    {
     public:
       <成员函数和数据成员的说明或实现>
     private:
       <数据成员和成员函数的说明或实现>
    };

//实现部分
    <函数类型> <类名>:: <成员函数名> (<参数表>)
    {
       <函数体>
    }
```

边界

特定的访问权限

计算机系 江志英 jiangzy@mail.buct.edu.cn

定义时钟类

```
class  Clock
{
 public:
      void SetTime(int NewH, int NewM,  int NewS);
      void ShowTime( );
 private:
      int Hour, Minute, Second;
};
```

成员函数

数据成员

计算机系 江志英 jiangzy@mail.buct.edu.cn

成员函数的实现

```
void  Clock::SetTime  (int NewH, int NewM, int NewS)
{
        Hour=NewH;
        Minute=NewM;
        Second=NewS;
}

void  Clock::ShowTime ( )
{
        cout<<Hour<<":"<<Minute<<":"<<Second;
}
```

*注意事项

1）在类体中不允许对数据成员进行初始化

```
class  Clock
{
 public:
     void SetTime(int NewH, int NewM,  int NewS);
     void ShowTime( );

 private:
     int Hour(10), Minute(30), Second(45);
};
```

*注意事项

2）类中数据成员的数据类型是没有限制的

- 基本数据类型
- 构造数据类型
- 指针和引用

数据成员的类型决定了该成员在内存中应占的字节数，同时也确定了该成员所允许的合法操作。

3）在类体内先说明公有成员，后说明私有成员；数据成员按类型大小，由小到大说明。

4）类的说明部分放到*.h文件，实现部分放在cpp文件

2 对象的定义

类的对象是该类的某一特定实体，即类类型的变量。

◆对象的定义格式

先定义类类型，再定义对象

<类名> <对象名表>;

例：

```
Clock  myClock，*pC, clocks[30];
Clock  &cl = myClock;
```


计算机系 江志英 jiangzy@mail.buct.edu.cn

◆对象成员的表示方法

（1）一般对象的成员表示用运算符“.”

<对象名>.<数据成员名>

<对象名>.<成员函数名>（<参数表>）

Clock myClock;

myClock.Hour ✕

myClock.SetTime(10,12,14)

（2）指向对象指针的成员表示用运算符“→”

<对象指针名>→<数据成员名>

<对象指针名>→<成员函数名>（<参数表>）

Clock *pC = &myClock;

pC→Hour ✗

pC→ SetTime(10,12,14)

（3）对象引用的成员表示用运算符“.”

<对象引用名>.<数据成员名>

<对象引用名>.<成员函数名>（<参数表>）

Clock &cl = myClock;

cl.Hour ×

cl.SetTime(10,12,14)

（4）对象数组元素的成员表示同一般对象

<数组名>[<下标>].<成员名>

Clock cl [3];

cl[0].Hour ×

cl[0].SetTime(10,12,14)

例

```
#include <iostream.h>
class Date
{
public:
        void  SetDate(int y, int m, int d)
        {
                year = y, month = m; day = d;
        }
        int  IsLeapYear()
        {
            return year%4==0&&year%100!=0||year%400==0;
        }
        void  Print()
        {    cout<<year<< '/' <<month<< '/' <<day<<endl;
        }
private:
        int year, month, day;
}
```


计算机系 江志英 jiangzy@mail.buct.edu.cn


```
void main()
{
        Date  d1, d2, *pd = &d2;
        d1.SetDate(2005, 6, 24);
        pd->SetDate(2000, 2, 8);
        cout << d1.IsLeapYear() <<   ‘,’   << d2.IsLeapYear() << endl;
        d1.Print();
        d2.Print();
}
```


计算机系 江志英 jiangzy@mail.buct.edu.cn

◆构造函数和析构函数

- 构造函数的功能是在创建对象时，用给定的值对对象进行初始化。
- 析构函数的功能是用来释放一个对象。
- 在这两个特殊函数由系统自动调用。


```
class Clock
{
public:
    Clock(int NewH,int NewM,int NewS); //构造函
    数
   ~Clock(); //构造函数
    void SetTime(int NewH,int NewM,int NewS);
    void ShowTime();

private:
    int Hour,Minute,Second;
};
```

•构造函数的实现：

```
Clock::Clock(int NewH, int NewM, int NewS)
{
        Hour= NewH;
        Minute= NewM;
        Second= NewS;
        cout<< “Constructor called.\n” ;
}
Clock::~Clock()
{
        cout<< “Desturctor called.\n” ;
}
```

•建立对象时构造函数的作用：

```
int main()
{
        Clock c(0,0,0); //隐含调用构造函数，将初始值作为实参
        c.ShowTime();
}
```

◆ **构造函数的特点**

① 构造函数的名字同类名。

② 说明或定义构造函数时不必指出类型，也无任何返回值。

③ 构造函数是系统在创建对象时自动调用的。

◆ **析构函数的特点**

① 析构函数名同类名，为与构造函数区别在析构函数名前加“～”符号。

② 析构函数定义时不必给出类型，也无返回值，并且无参数。

③ 析构函数是由系统自动调用。

析构函数由于没有参数，它不能被重载。

```
#include<iostream>
using namespace std;
class Point
{
 public:
        Point(int xx,int yy);
        ~Point();
        //...其他函数原型

 private:
        int X,int Y;
};
```

```
Point::Point(int xx,int yy)
{
        X=xx;
        Y=yy;
}
Point::~Point()
{ }
//...其他函数的实现略
```


计算机系 江志英 jiangzy@mail.buct.edu.cn

◆默认构造函数和默认析构函数

- 默认构造函数特点是不带参数
  - 系统自动提供一个默认形式的构造函数
  - 程序员定义的默认构造函数

Date d1,d2

- 默认析构函数，系统提供的

◆拷贝构造函数

■用已知对象初始化创建另一对象时所用的构造函数。

```
class 类名
{
    public :
        类名（形参）；//构造函数
        类名（类名 &对象名）；//拷贝构造函数
            ...
};
类名::类名（类名 &对象名）//拷贝构造函数的实现
{    函数体    }
```

如果没有定义拷贝构造函数，系统自动创建一个默认


```
class Point
{
  public:
     Point(int xx=0,int yy=0){X=xx; Y=yy;}
     Point(Point&  p);
     int GetX() {return X;}
     int GetY() {return Y;}
  private:
     int  X,Y;
};
Point::Point (Point&  p)
{
    X=p.X;          Y=p.Y;
    cout<<"拷贝构造函数被调用"<<endl;
}
```

◆拷贝构造函数的调用

当用类的一个对象去初始化该类的另一个对象时系统自动调用拷贝构造函数实现拷贝赋值。

```
int main()
{
    Point A(1,2);
    Point B(A); //拷贝构造函数被调用
    cout<<B.GetX()<<endl;
}
```

◆拷贝构造函数

若函数的形参为类对象，调用函数时，实参赋值给形参，系统自动调用拷贝构造函数。例如：

```
void fun1(Point  p)
{
    cout<<p.GetX()<<endl;
}
int main()
{
    Point A(1,2);
    fun1(A); //调用拷贝构造函数
}
```

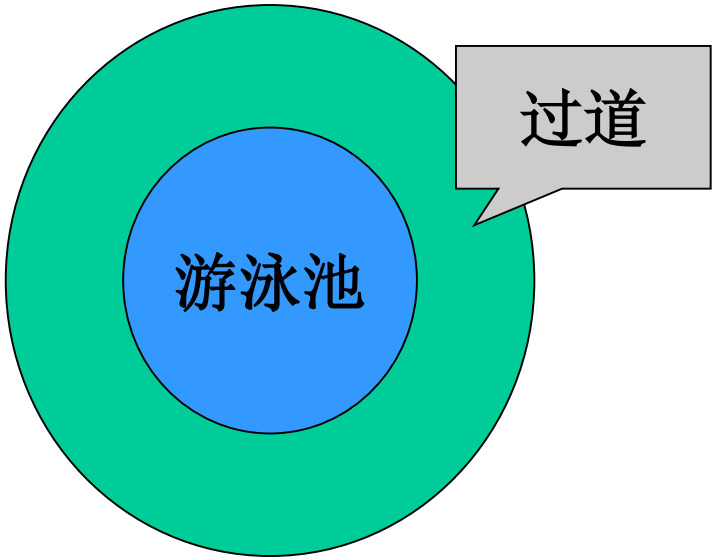
当函数的返回值是类对象时，系统自动调用拷贝构造函数。

例如：

```
Point fun2()
{   Point A(1,2);
    return   A; //调用拷贝构造函数
}
int main()
{
    Point B;
    B = fun2();
}
```

类的应用举例

一圆形游泳池如图所示，现在需在其周围建一圆形过道，并在其四周围上栅栏。栅栏价格为35元/米，过道造价为20元/平方米。过道宽度为3米，游泳池半径由键盘输入。要求编程计算并输出过道和栅栏的造价。




```
#include <iostream>
using namespace std;
const  float  PI  = 3.14159;
const  float  FencePrice  = 35;
const  float  ConcretePrice  = 20;

//声明类Circle 及其数据和方法
class Circle
{
 private:
   float   radius;

 public:
   Circle(float r);  //构造函数

   float  Circumference( ) const; //圆周长
   float  Area( ) const;  //圆面积
};
```


```
// 类的实现
// 构造函数初始化数据成员radius
Circle::Circle(float r)
{       radius=r        }

// 计算圆的周长
float Circle::Circumference( ) const
{
        return 2 * PI * radius;
}

// 计算圆的面积
float Circle::Area( ) const
{
        return PI * radius * radius;
}
```

```
void main ()  {
        float radius;
        float FenceCost, ConcreteCost;
        // 提示用户输入半径
        cout<<"Enter the radius of the pool: ";
        cin>>radius;
        Circle Pool(radius);        // 声明 Circle 对象
        Circle PoolRim(radius + 3);
        //计算栅栏造价并输出
        FenceCost=PoolRim.Circumference()*FencePrice;
        cout<<"Fencing Cost is ￥"<<FenceCost<<endl;

        //计算过道造价并输出
        ConcreteCost=(PoolRim.Area()  -
                        Pool.Area())*ConcretePrice;
        cout<<"Concrete Cost is ￥"<<ConcreteCost<<endl;
}
```

4 成员函数的特性

- 内联函数
- 重载性
- 设置参数的默认值

◆ 内联函数

- 内联函数是一种函数体被替换，而不是被调用的函数。（提高运行时的效率）
- 内联函数体中不要有复杂结构（如循环语句和switch语句）。
- 在类中声明内联成员函数的方式：
  - 将函数体放在类的声明中。
  - 使用inline关键字。


```
#include <iostream.h>
  class M
  {
   public:
     M(int i,int j)
     {   a=i;b=j; }
     int fun1()
     {  return a;  }
     int fun2()
     {  return b;  }
     int fun3(),fun4();
   private:
     int a,b;
  };
```

```
inline int M::fun3()
 {
    return fun1()+fun2();
 }
 inline int M::fun4()
 {
    return fun3();
 }

 void main()
 {
    M m(5,8);
    int n=m.fun4();
    cout<<n<<endl;
 }
```

◆ 重载函数

成员函数可以重载，重载时应遵循参数可以区别的规则。

- 参数的类型
- 参数的个数

```
#include <iostream.h>
  class AB
  {
   public:
        AB(int i,int j)      { a=i;b=j; }
        AB(int i)            {  a=i; b=i*i;  }
        int Add(int x, int y);
        int Add(int x);
        int Add();
        int aout()     {  return a;  }
        int bout()     {  return b;  }
   private:
        int a,b;
  };
```

```
int AB::Add(int x,int y)
  {
     a=x;
     b=y;
     return a+b;
  }
  int AB::Add(int x)
  {
     a=b=x;
     return a+b;
  }
  int AB::Add()
  {
     return a+b;
  }
```


```
void main()
  {
        AB a(5,8),b(7);
        cout<<"a="<<a.aout()<<','<<a.bout()<<endl;
        cout<<"b="<<b.aout()<<','<<b.bout()<<endl;

        int i=a.Add();
        int j=a.Add(4);
        int k=b.Add(3,9);
        cout<<i<<endl<<j<<endl<<k<<endl;
  }
```


计算机系 江志英 jiangzy@mail.buct.edu.cn

◆ 设置参数的默认值

- 成员函数的参数可以设置为默认值。
- 可以给一个或多个参数设置默认值;
- 指定了默认值的参数的右边，不能出现没有指定默认值的参数。

A(int i=8, int j=10, int k);　　错

会影响重载

```
#include <iostream.h>
  class A
  {
   public:
     A(int i=8, int j=10, int k=12);
     int aout()   {  return a;  }
     int bout()   {  return b;  }
     int cout()   {  return c;  }
   private:
     int a,b,c;
  };
A::A(int i,int j,int k)
 {
    a=i; b=j; c=k;
 }
void main( )
   {
     A X,Y(5),Z(7,9,11);
     cout<<"X="<<X.aout()<<','<<X.bout()<<','<<X.cout()<<endl;
     cout<<"Y="<<Y.aout()<<','<<Y.bout()<<','<<Y.cout()<<endl;
     cout<<"Z="<<Z.aout()<<','<<Z.bout()<<','<<Z.cout()<<endl;
   }
```

5 静态成员

- 静态成员解决了数据共享的问题。
- 在类体内使用关键字static说明的成员称为静态成员，包括：
  - 静态数据成员
  - 静态成员函数
- 静态成员的特点是它不是属于某对象的，而是属于整个类的，即所有对象的。
- 静态成员的访问，可以通过对象引用，也可以通过类名来引用。

◆ 静态数据成员

1．静态数据成员的说明方法

定义静态数据成员的语句格式如下：

```
class M
{
   int  a, b, c;
   static int   s;
   ……
};
```


计算机系 江志英 jiangzy@mail.buct.edu.cn

2．静态数据成员的初始化及访问方式

在类体外初始化：

<数据类型> <类名>::<数据成员名>=<初值>;

int M::s=0;

访问方式：

<类名>::<静态数据成员名>;

M::s

```
#include <iostream.h>
  class MY
  {
   public:
     MY(int i,int j,int k);
     void PrintNumber();
     int GetSum(MY m);
   private:
     int a,b,c;
     static int s;
  };
  int MY::s=0;
```

在全局区域初始化

3. 静态数据成员被存放在静态存储区

必须初始化；

静态数据成员所占的空间不会随着对象的产生而分配，也不会随着对象的消失而回收；

只有在程序结束时才被系统释放。

```
#include <iostream.h>
class MY {
  public:
    MY(int i,int j,int k);
    void PrintNumber();
    int GetSum(MY m);
  private:
    int a,b,c;
    static int s;
};
int MY::s=0;

MY::MY(int i,int j,int k)
{
   a=i;b=j;c=k;
   s=a+b+c;
}
void MY::PrintNumber()
{
   cout<<a<<','<<b<<',  '
                <<c<<endl;
}
int MY::GetSum(MY m)
{
   return MY::s;
}

void main( ) {
   MY m1(2,3,4),m2(5,6,7);
   m2.PrintNumber();
   cout<<m1.GetSum(m1)<<','<<m2.GetSum(m2)<<endl;
}
```

◆ 静态成员函数

- 静态成员函数格式如下：

  static <类型> <成员函数名>(<参数表>);

- 引用静态成员函数有如下两种方式：
  - <类名>::<静态成员函数名>(<参数表>)
  - <对象名>.<静态成员函数名>(<参数表>)

- 在静态成员函数中可以直接引用其静态成员，而引用非静态成员时需用对象名引用。

```
#include <iostream.h>
#include <string.h>
class Student {
  public:
    Student(char name1[],int sco)
        {       strcpy(name,name1);     score=sco;        }
    void total( )
        {       sum+=score;             count++;          }
    static double aver( )
        {       return (double) sum/count;                }
  private:
    char name[20];
    int score;
    static int sum, count;
};
```


```
int  Student::sum = 0;
int  Student::count = 0;  // 对静态成员变量进行初始化

void main()
{
   Student stu[5] =  {
        Student("Ma",89),  Student("Hu",90),  Student("LU",95),
        Student("Li",88),  Student("Gao",75)
   };

  for(int i=0;i<5;i++)
        stu[i].total();

  cout<<"Average="<<Student::aver()<<endl;
}
```

静态成员函数举例

```
class A
{
   public:
      static void f(A a);  // 静态成员函数
   private:
      int x;  // 非静态成员变量
};
void A::f(A a)
{
   cout << x ;              //对x的引用是错误的
   cout << a.x ;            //正确
}
```

具有静态成员的 Point类

```
#include <iostream.h>
class Point {      //Point类声明
    public:        //外部接口
        Point(int xx=0,int yy=0){X=xx;Y=yy;countP++;}
        Point(Point &p);//拷贝构造函数
        int GetX() {return X;}
        int GetY() {return Y;}
        static void GetC( ) { cout<<" Object id="<<countP<<endl; }

     private: //私有数据成员
        int   X,Y;
        static  int   countP;
};
```


计算机系 江志英 jiangzy@mail.buct.edu.cn


```
Point::Point(Point &p) {
        X=p.X;
        Y=p.Y;
        countP++;
}
int Point::countP=0;
int main( ) {     //主函数实现
        Point A(4,5);     //声明对象A
        cout<<"Point A,"<<A.GetX()<<","<<A.GetY();
        A.GetC();         //输出对象号，对象名引用
        Point B(A);       //声明对象B
        cout<<"Point B,"<<B.GetX()<<","<<B.GetY();
        Point::GetC();    //输出对象号，类名引用
}
```

6 友元

- 友元是C++提供的一种破坏数据封装和数据隐藏的机制。

- 通过将一个模块声明为另一个模块的友元，一个模块能够引用到另一个模块中本是被隐藏的信息。

- 可以使用友元函数和友元类。

- 为了确保数据的完整性，及数据封装与隐藏的原则，建议尽量不使用或少使用友元。

◆ 友元函数

友元函数是说明在类体内的一般函数，它不是这个类中的成员函数，但是它访问该类所有成员。

友元函数说明格式如下：

friend <类型> <函数名>（<参数表>）

使用友元函数时应注意如下几点：

① 友元函数前边加 friend 关键字，说明在类体内。如被定义在类体外，不加类名限定。

② 友元函数可以访问类中的私有成员和其他成员。

③ 友元函数的作用在于可以提高程序的运行效率。

④ 友元函数在调用上同一般函数。


```
#include <math.h>
class Point {
  public:
        Point(double i,double j)
        {       x=i;      y=j;      }
        void Print()
        {       cout<<'('<<x<<','<<y<<')'<<endl;  }
        friend  double  Distance( Point a, Point b ); //友元声明
  private:
       double x,y;
};

double Distance ( Point a, Point b ) {  // 类体外定义，无需类名
   double dx = a.x - b.x;
   double dy = a.y - b.y;
   return sqrt( dx*dx + dy*dy );
}
```


```
void main( )
  {
        double d1=3, d2=4, d3=6, d4=8;
        Point  p1( d1, d2 ), p2( d3, d4 );
        p1.Print( );
        p2.Print( );
        double d = Distance( p1, p2 );
                  // 友元调用，同普通函数
        cout<<"Distance="<<d<<endl;
  }
```

◆ 友元类

若一个类为另一个类的友元，则此类的所有成员都能访问对方类的私有成员。

声明语法：

将友元类名在另一个类中使用friend修饰说明。

```
friend  class  <类名>;
```

使用友元类应注意下述事项：

① 友元关系是不可逆的：

B类是A类的友元类，不等于A类是B类的友元类。

② 友元关系是不可传递的：

B类是A类的友元类，C类是B类的友元类，C类不一定就是A类的友元类。


```
#include <iostream.h>
class X {
    friend class Y;
    public:
        X(int i) {  x=i;  }
        void Print()  {
            cout<<"x="<<x<<',  '
                <<"s="<<s<<endl;  }
    private:
        int x;
        static int s;
    };
int X::s=5;
class Y {
    public:
        Y(int i) {  y=i;  }
        void Print(X &r) {
            cout<<"x="<<r.x<<',  '
                <<"y="<<y<<endl;  }
    private:
        int y;
    };

void main() {
        X m( 2 );
        m.Print( ) ;
        Y n( 8 );
        n.Print( m );
}
```


计算机系 江志英 jiangzy@mail.buct.edu.cn

本章小结

- 类的定义和对象的定义
- 构造函数和析构函数
- 成员函数的特性
- 静态成员
- 友元